Control Demonstration of a Thin Deformable In-Plane Actuated Mirror

THESIS

Gina A. Peterson, Captain, USAF

AFIT/GSS/ENY/06-M10

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GSS/ENY/06-M10

Control Demonstration of a Thin Deformable In-Plane Actuated Mirror

THESIS

Presented to the Faculty

Department of Aeronautical and Astronautical Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science (Space Systems)

Gina A. Peterson, B.S.E.

Captain, USAF

March 2006

AFIT/GSS/ENY/06-M10

Control Demonstration of a Thin Deformable In-Plane Actuated Mirror

Gina A. Peterson, B.S.E.

Captain, USAF

Approved:

/signed/	21 Mar 2006
Dr. Richard G. Cobb Chairman	date
/signed/	21 Mar 2006
Dr. Anthony N. Palazotto Member	date
/signed/	21 Mar 2006
Dr. Donald L. Kunz Member	date

Abstract

Current imaging satellites are limited in resolution and field of regard by the aperture size of their primary optical mirror. To get a large optical mirror into space, current launch weight and size restrictions must be overcome. Membrane-like optical mirrors can overcome these restrictions with their very lightweight and flexible properties. However, thin, deformable membrane mirrors are very susceptible to the space environment and require active control for surface stabilization and shaping. The primary goal of this research is to demonstrate that an in-plane actuated membrane-like deformable optical mirror can be controlled to optical wavelength tolerances in a closed-loop system. Fabrication and characterization of a five-inch membrane-like optical mirror is carried out based on efforts made in previous research. A data acquisition system to implement closed-loop feedback is characterized. Validity of a closed-loop control method is demonstrated with the in-plane actuated deformable mirror.

Acknowledgements

I would like to give a special thanks to Dr. Cobb (AFIT/ENY), Dr. Mollenhauer (AFRL/ML), Maj. Shepherd (AFIT/ENY), Jay Anderson, and Wilber Lacy. I would also like to acknowledge the contribution of Andrew Marcum an electrical engineering student at Rose-Hulman Institute of Technology, without his hard work and ASCII knowledge, this work would have been an impossible challenge. Finally, I'd like to thank my husband for all his love and support.

Gina A. Peterson

Table of Contents

Page

Page

List of Figures

List of Tables

Control Demonstration of a Thin Deformable In-Plane Actuated Mirror

I. Introduction

1.1 Overview

Continuous global communication and surveillance of all adversaries in radar, visual, infrared (IR), and microwave wavelengths is an integral part to achieving one of the U.S. Air Forces distinctive capabilities of information superiority. One of many examples of information superiority was the US timely response to the October 1994 Iraqi force build-up that threatened Kuwait [11]. The information operation, information-in-warfare (IIW), provides commanders with total battle space situational awareness through intelligence, surveillance, and reconnaissance (ISR). Currently, global surveillance is limited in resolution and field of regard by the aperture size of a satellite's imaging collecting area, or its primary optical mirror. The field or regard is increased with a constellation of satellites around the earth but resolution is not. To increase the field of regard per imaging satellite and achieve better resolution of earth objects, the aperture size of the primary optical mirror must be increased [35].

Over the past several decades, the scientific community has researched ways to get large optical mirrors in space because from space, telescopes can avoid atmospheric aberrations, atmosphere absorption (including some IR wavelengths beyond 2 μm), and thermal emission, which interferes with detection of faint objects [2]. Similar to earth surveillance satellites, the optical aperture size of telescopes used to observe space must increase to achieve better resolution and field of regard. In general, the payload capacity of launch vehicles is the main limitation to enlarging current space mirror designs. Therefore, research efforts to get large mirrors in space have concentrated on reducing launch weight and launch size.

To meet launch weight restrictions, the areal density of the mirror must decrease as its size increases. The areal density is the total weight divided by the collecting aperture area. The Hubble Space Telescope (HST) 2.4 m primary mirror is made of ultra-low expansion (ULE) glass and has an areal density of 180 $\frac{kg}{m^2}$ [7]. It is not feasible to enlarge a HST type mirror because it would be too heavy to get to space. Research dedicated to reduce areal density to less than 25 $\frac{kg}{m^2}$ coined the term *lightweight optical mirrors*. Besides low-density materials, these mirrors are developed with unique structures to reduce material bulk while maintaining their shape. Several lightweight, high strength materials such as ULE, Zeodur, beryllium, NT-SiC, CESIC, and graphite-epoxy are being studied for this type of mirror [6], [36], [14], and [22]. An alternative to mirrors is a refractive, or

Fresnel, lens that requires a fraction of the volume of transparent material to focus light compared to a reflective lens. This option reduces areal density significantly but requires redesign of telescope structures [17].

The trend to further reduce areal density is supported by current and future space mirror size requirements. In 1999, the Department of Defense (DoD) and National Aeronautical Space Administrations (NASA) Advanced Mirror System Demonstrator (AMSD) program required an areal density less than 15 $\frac{kg}{m^2}$ for near future space telescopes [25]. The result is a beryllium mirror developed for the James Webb Space Telescope to launch in June 2013. The 6.5 m mirror will have an areal density of 19.3 $\frac{kg}{m^2}$ [30]. Concepts for NASAs future 40-60 m Gossamer telescope require areal densities less then 1 $\frac{kg}{m^2}$ [40].

Large mirrors must also meet launch size restrictions. Currently, the solution is to divide the mirror into multiple pieces. This is achieved by putting together many small mirror segments to make one big mirror. The JWST will have eighteen 1.3 m mirror segments to make the primary mirror. A second way this is achieved is to have several small telescopes fly in formation to create a large mirror effect. The Terrestrial Planet Finder (TPF) observatory is a concept that requires formation flying of several small telescopes to create a large aperture for mid-IR interferometry. A future solution for gossamer size mirrors suggested by NASA is to make the mirror out of a collapsible fabric that can be rolled or folded up during launch and then deployed in space [12]. A large inflatable deployable antenna flight in May of 1996 demonstrated that thin collapsible reflectors were plausible in space [24]. Achieving an optical quality collapsible surface in space is the next step.

Research in thin, deformable, membrane mirrors suggests they are candidates to meet launch restrictions and future gossamer size requirements. In general, a membrane is very lightweight and could be rolled or folded. The main challenges remain to first create a membrane mirror with an optical quality surface, and second maintain the shape and surface tolerances to optical level in the space environment. Unlike traditional rigid mirrors and lightweight semi-rigid mirrors, some mechanism of actuation and control is needed to steady the mirror. Once control is achieved, a deformable membrane mirror could correct for changes in the space environment, material property changes over time, and initial shape deformations [9].

The most common type of deformable mirrors have a support, or reaction structure that holds in place actuators either behind and/or at the edges of the mirror. Figure 1 shows a conventional deformable mirror structure. The back-plate provides rigidity, the faceplate is the reflector, and the reaction structure contains the actuators. Although the deformable mirror is very lightweight, the actuators and reaction structure increase the overall weight and areal density of a mirror by up to 50 percent [19]. For example, if the JWST mirror did not require a support structure, the

areal density reduces to 13.4 $\frac{kg}{m^2}$ [12]. Three main solutions have been suggested to reduce support structure weight. One solution is to reduce the number of actuators by optimizing their placement. However, a fail-safe system has more actuators than are necessary, because if one or two fail it will not significantly affect the mirror shape [9]. The second solution reduces weight by only using outer-edge actuators. Outer-edge actuators provide sufficient shape control but are best used along with surface actuators to increase the fidelity of control. A third solution to use embedded actuators virtually eliminates support structure weight. Embedded actuators add no significant weight and do not require an external support structure to react against.

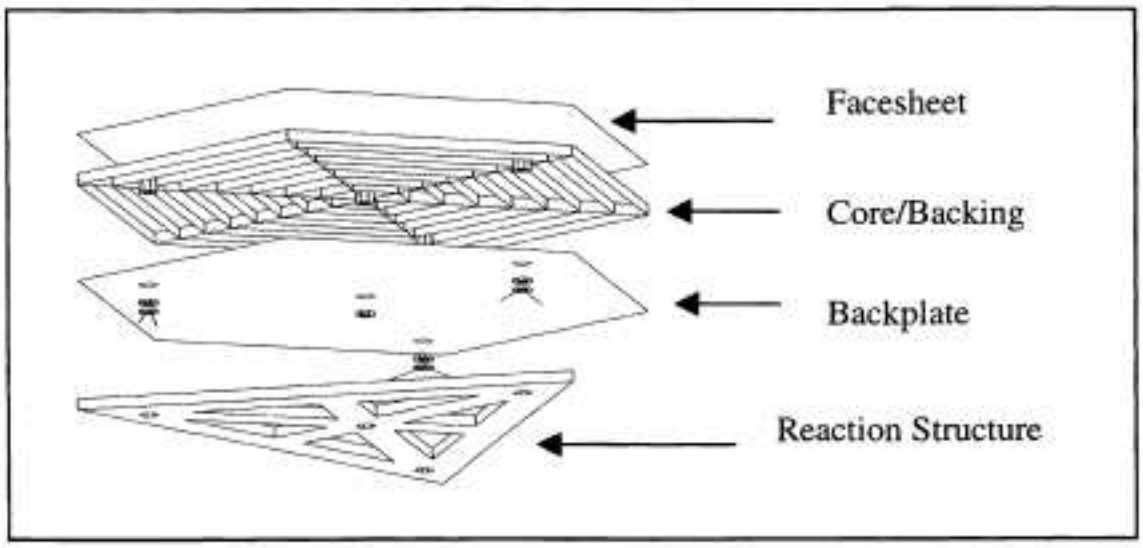

Figure 1: Typical lightweight mirror design [25]

A gossamer size optical-quality space mirror may be achievable with a thin, in-plane actuated, deformable, membrane mirror. Challenges to achieve a local surface optical quality flatness of a few microns and maintain global shape in a changing space environment are extreme. A good system design will also include a method to self-control through on-board sensors. The direction of space exploration motivates research in all areas to achieve such a feat. Once deployed, advancements in ISR and other military applications could be staggering.

1.2 Problem

The primary focus of this research is to perform an experimental demonstration using quasi-static closed-loop feedback on an in-plane actuated deformable membrane mirror. Previous work at the Air Force Institute of Technology (AFIT) demonstrated open-loop optical quality deformations on an in-plane actuated deformable mirror. Enough deflection to correct for local surface defor-

mations in an optical surface was achieved. However, demonstrating that active control can damp out low frequency disturbances is a necessary step to demonstrate potential for future space applications. A well-documented deformable mirror closed-loop method of control that uses actuator influence functions is desired to demonstrate the mirror's surface feedback control capability. A data acquisition system that can implement closed-loop feedback must first be created using Simulink® and dSpace® control hardware and software.

1.3 Scope

The two main objectives of this research are concerned with demonstrating quasi-static control in an in-plane actuated deformable membrane mirror. The first objective is to set-up an acquisition system that can provide feedback control at a rate exceeding 3 Hz. Previous research at the AFIT has characterized static actuation (up to two Hz) of a five-inch in-plane actuated deformable mirror through data collection. Characterization was limited by measurement rate, data acquisition rate, and the inability to provide feedback. Previously, a Shack-Hartmann type wavefront sensor manufactured by Adaptive Optics Associates (AoA) provided surface deformation data and a data acquisition methodology was developed in Matlab® to capture it. The measurement rate is increased with an upgrade in AoA hardware and software. Receiving data with Simulink® and dSpace® increases the acquisition sample rate and provides a means to send feedback to the mirror.

The second objective is to demonstrate quasi-static control feedback at rates as fast as the hardware will allow. Ultimately a least-squares control methodology will be implemented. Quasi-static control will be demonstrated by controlling the surface to an achievable desired static shape through closed-loop feedback. This will characterize the mirrors ability to correct for manufacturing imperfections. A low frequency disturbance will then be introduced into the loop. This will simulate the mirror's ability to correct for low frequency space disturbances such as thermally-induced distortions of the mirror's surface.

1.4 Summary of Thesis

This thesis discusses related research in this area and presents the test results to control a thin, piezoelectric, deformable membrane mirror with embedded actuators.

The second chapter reviews research in different methods of actuation and control of deformable membrane mirrors. This short review illustrates the advancement and diversity of attempts to achieve a large thin mirror. The third chapter details the data acquisition set-up, including three Simulink® models that were developed to implement it. Chapter Three also describes the control algorithm and

presents results of open-loop control experiments. Chapter Four presents the results of closed-loop control experiments and introducing a low-frequency disturbance into the system. Chapter Five outlines all conclusions and recommendations for future research.

II. Review of the Relevant Literature

2.1 Overview

Many research studies to produce and characterize deformable membrane mirrors aid efforts to determine the feasibility of a flexible optical mirror in space. Most significant is the development of membrane materials that can produce an optical reflective surface and retain their flexible, operational, and optical surface tolerances in space.

First, membrane materials must retain flexibility even with active materials attached for control purposes. This is especially important for space applications because a large deformable mirror will need to be compactly packaged for launch. An extensive study on the effects of rolling and unrolling a hyper-thin composite mirror developed a measure of the maximum rolling hysteresis that could be corrected by the mirror [29]. This measurement analysis suggests rolling effects can be corrected.

Second, active membrane materials must retain their physical properties once deployed in space. In space, the effects of ionizing radiation, atomic oxygen exposure, and thermal cycling must be considered. One study determined the effects of temperature cycling in a low earth orbit on polyvinylidene fluoride (PVDF). PVDF is often used as an active material to shape membrane mirrors. The PVDF materials polarization stability and ability to deform were significantly affected [13]. This type of study is very specific to each manufactured mirror. Similar tests could be done for future candidate space membrane mirrors.

Finally, concerns on achieving and maintaining optical tolerances from highly flexible surfaces exist for ground and space applications. Since a membrane mirror is very susceptible to the space environment, a closed-loop control system is needed. Two pieces of the system include method of actuation and a control model. An overview of research in these areas follows.

2.2 Actuation Methods

Although the lightweight, flexible nature of membrane mirrors is ideal to meet launch restrictions, it also makes them susceptible to distortions caused from manufacturing, deployment, and the space environment. Surface precision of an optical quality mirror is on the order of $\frac{\lambda}{8}$ to $\frac{\lambda}{20}$ depending on project requirements [4], [29]. This is about an order of magnitude better than the wavelength of the light to be reflected. For visible light, a surface quality around 50 nm is required. Some type of control is needed to maintain surface shape and tolerances. Various techniques to apply active control through a moment or force by actuators on the mirror have been proposed. An overview of the advancements in proposed actuation methods follows as a basis of comparison in this research.

2.2.1 Global Surface Shape Control. The simplest fix is a passive method that stiffens the membrane to a desired shape. Rigidizing a membrane mimics traditional glass mirrors that cannot

be deformed by the forces in a space environment. An early method to stiffen the membrane was to use a pre-shaped inflatable structure. The Air Force Research Laboratory (AFRL) and the South Dakota School of Mines and Technology proved a better passive method. They found that applying a coating that exactly counteracts the resin curing stress of a membrane would preserve the surface and shape of its mold. This technique assumes the membrane mirror is manufactured from a highly polished mold equal to the size of the mirror to be made [4].

Overall shape control is possible through boundary actuation. This was demonstrated in simulation with the boundary manipulation of inflated structures [21]. It has also been demonstrated for membranes that are stretched over their frames. One stretched membrane experiment uses rails that are actuated by forces and moments applied to their ends [1]. Another stretched membrane experiment uses a lightweight frame with a discrete number of edge nodes. Electrostatic forces on the nodes are used to control the shape [2].

Finally, a distributed voltage across a flat piezo-active polymer can be used to deform it to a desired shape. Researchers at NASA have verified this computationally [5]. In general, global-shape methods may be best used with another control method in order to increase the control capability to include reacting to space environment changes.

2.2.2 Local Surface Tolerance Control. Conventional deformable mirrors have external reaction structures and actuators as shown in Figure 1. A membrane rests on a bed of actuators in this type of design. Only the number of actuators and their deflection capability limits local surface control. In general, this design is envisioned for segmented space mirrors or mirrors developed for ground telescopes because of the additional weight. This type of design is also being studied to replace the inflated canopies for space antennas. An experiment was designed where a four-quadrant electrode grid sitting behind a pre-shaped membrane mirror uses electrostatic forces to deform the surface. Any manufacturing distortions can be removed by this method [15].

Internal actuators are characterized by not requiring any reaction structure and offer the most low areal density application potential for active control of membrane mirrors. The reflective surface is also the reactive structure in this design because the force or moment is introduced in the plane of the reflective surface. This is accomplished as easily as attaching polyvinylidene fluoride (PVDF) strips to a membrane mirror [38]. Two very pertinent studies have been completed by the Jet Propulsion Laboratory at the California Institute of Technology and by AFIT.

In California, the concept of applying piezoelectric ceramic strip actuators to a composite lightweight mirror for the correction of long-wave lower order distortions has been demonstrated successfully by analysis and experiment. Here, a long-wave lower order distortion, or a *warping*

distortion, is caused by small nonhomogeneity or nonsymmetry in the mirror. The actuators induce strains to deform the front face. Thirty strips on a 0.5 m diameter, 2.5 cm thick, mirror along 6 equally spaced radii demonstrated correction for errors in focus, astigmatism, spherical aberration, and coma. This study concluded that the optimal application of their concept would be to have a composite mirror that has a back face sheet made of piezoelectric material with a matrix of printed circuitry [23].

A similar study was completed at the Swiss Federal Institute of Technology to successfully correct for gravity distortions. This study actuated 12 piezo-ceramic patches attached to a fiber-reinforced composite mirror. The RMS-error due to gravity was reduced in the mirror by more than 80 percent [27].

Research at AFIT has used piezoelectric material with in-plane actuators to deform a 5-inch membrane mirror. The mirror is a composite strucutre with an inert silicone substrate and active PVDF layer. In this case, deformations up to 3.15 μm have been demonstrated in the surface. This method produced mirrors with areal densities less then 4 $\frac{kg}{m^2}$ [35].

2.3 Control Algorithms

Most studies on how to best implement the actuators on a general deformable mirror suggest an algorithm that utilizes influence functions (IFS). Actuator IFS describe the deformed surface shape that results from a force applied by the actuators. To implement closed-loop control, a controller matrix based on the knowledge of the IFS is developed. Measured wavefront error is corrected with actuator commands determined by the controller matrix in a closed feedback loop. One experiment that used a control matrix derived from experimentally measured IFS demonstrates that this is a valid technique. The 97-actuator conventional deformable mirror system achieved a 50 percent improvement in surface quality through an iterative process where new actuator commands were determined from the measured wavefront error and control matrix [34]. There are many studies on how analytically derived IFS compare to each other. An overview of a few of these studies emphasizes the uniqueness of each derivation and comparison.

Each derivation starts with a model of the mirror. One approach assumes each actuator is a point force and models each as a spring attached to a rigid reaction structure. In one study, the actuator IFS are analytically derived for a circular plate by a Fourier series expansion. They are found to be similar to traditional finite-element analysis [26]. In a similar way, the influence function shapes from the finite element model are compared to experimentally measured shapes. For model simulation, analytically determined shapes reduced execution times significantly and

eliminate peculiarities that show up in the actual surface [20]. Another derivation with the same initial model determines uniform-load IFS only for circular mirrors with a hole in the center [3].

A more complex analytic derivation achieves a nonlinear feedback controller by first determining nonlinear IFS. The modal non-interaction control represents the ability to control the amplitude of individual modes as long as the displacement at each actuator can be measured. The derivation uses a linear approximation when surface displacements are much less then the mirror thickness. This derivation is for circular, deformable, electrostatic, membrane mirrors with modal representation by Zernike polynomials [37].

There are also other derivations that don't use IFS. One example utilizes multidimensional systems theory. The conceptual control algorithm assumes a future structure with built-in actuators, sensors, and computational elements. It requires local error information and actuator capability knowledge. The theory suggests that closed loop control in space is easy as long as sensors and actuators are interchangeable [16].

An in-plane actuated deformable mirror that has been studied at AFIT has potential to demonstrate modal shape control and error correction. It is the goal of this research to implement IFS in a closed-loop feedback system successfully on this mirror. Once accomplished, further characterization and alternative control methods may be desired to demonstrate feasibility of this type mirror in space applications.

III. Data Acquisition System

3.1 Overview

This chapter describes the data acquisition system that was developed in order to perform closed-loop control experiments. The data acquisition system is described by giving an overview of each component in the system. This includes a description of the in-plane actuated deformable test mirror and the optical table set-up that has been used by previous AFIT researchers. It also includes an explanation of the data conversion between the components within the system. To capture the dynamics of the test mirror, a system operating rate of 492 Hz is needed. Converting the data slows down the system operating rate significantly. Therefore a discussion on the final data rate achieved by the system is included.

3.2 Data Sampling

The objective of the data acquisition system is to capture and control the test mirror. In general, closed-loop control works by taking a measurement of the test mirror surface and comparing it to a desired surface shape. If there is any difference between the two, the test mirror actuators are adjusted to correct it. This is repeated to maintain a desired shape in the surface of the mirror. This loop depends on being able to accurately capture the mirror's surface. The test mirror surface fluctuates due to air current and vibrational noise. The frequency of this noise is unknown, but its effect will show-up mostly in the first few modal frequencies of the mirror. This means that the mirror has significant fluctuations up to 246 Hz that need to be captured (see Section 3.3.2).

Sampling theory states that a continuous signal must be sampled at least twice as fast as all frequencies contained in the signal. If it is not, the sampled signal will not accurately represent the original continuous signal. This is called aliasing and is demonstrated in Figure 2. Here the solid line represents a 5 Hz sinusoid signal actuating the test mirror. The circles represent the mirror when it is sampled at 9.4 Hz (just shy of the 10 Hz suggested by sampling theory). The sampled signal has a period of 4.4 Hz if the circles are connected together. Therefore, the 5 Hz sinusoid is not accurately represented by this sampling rate.

This means that if the test mirror is not measured at a rate of at least 492 Hz, the mirror's surface movement will not be captured accurately and will instead appear to be moving much slower than it actually is. If the mirror's movement can not be seen, it can not be controlled. Therefore it is the goal of the data acquisition system to operate at a rate of 492 Hz. Increasing the rate above this is a misuse of memory space.

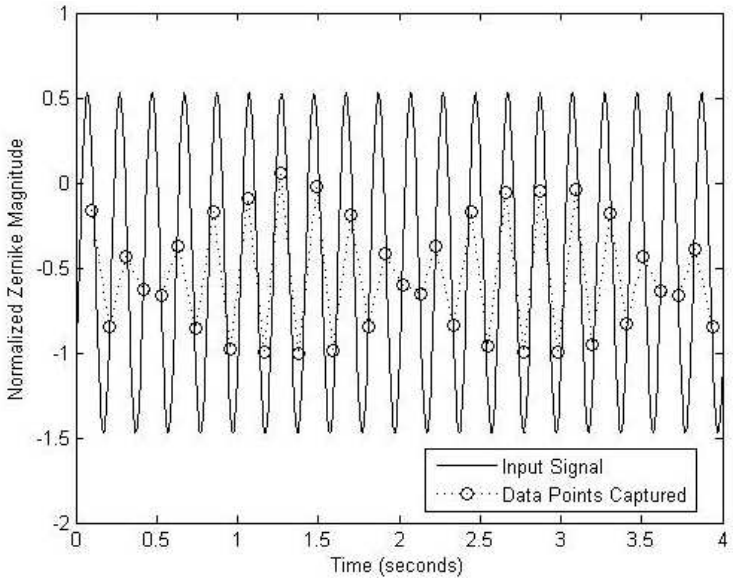

Figure 2: Example of Aliasing a Signal

3.3 Test Mirror

3.3.1 Description. The test mirror is a unimorph in-plane actuated tensioned membrane mirror. In summary, the mirror is a composite structure of a 1.2 mm room temperature vulcanizing (RTV) silicone substrate, with a near-optical coating of palladium gold on the order of several microns on the reflective side and a 52 μm PVDF actuating layer on the other.

The actuation layer consists of a pattern etched out of the copper-nickel layer on one side of the PVDF material. The pattern forms six radial actuators and one center axi-symmetric circular actuator as shown in Figure 3. Actuator pattern dimensions are documented by Trad [35]. The pattern was carefully aligned with a known direction of the PVDF. The PVDF material is non-orthotropic in nature as the strength of the piezoelectric coefficient in the y-direction is over seven times the strength of the coefficient in the x-direction (See Table 1). Notice that each actuator has thin leads running to the outer boundary where wires can be attached later. The PVDF actuators are capable of being energized singly or in combination with static and dynamic potentials up to plus and minus 600 volts.

The PVDF sheet is then clamped in tension and glued to a rigid 5-inch diameter circular aluminum ring using M-bond 600 epoxy. Every effort is made to uniformly tension the mirror. RTV silicone substrate is poured on the surface and allowed to cure. The palladium gold layer is then evaporated on to the silicone surface in a vacuum chamber. This produces a reflective surface as shown in Figure 4. The manufacturing procedure of the mirror was covered in detail by Sobers [32] [33].

Table 1: PVDF Piezo Film Properties

Parameter	Value	Units
thickness	52	μm (micron, 10^{-6} m)
d_{31}	3	$10^{-12}\ \frac{m}{V}$
d_{32}	23	$10^{-12}\ \frac{m}{V}$
Young's modulus	2-4	$10^{9}\ \frac{kg}{m^3}$
density	1780	$\frac{kg}{m^3}$
useful temperature range	-40^o to 100^o	C

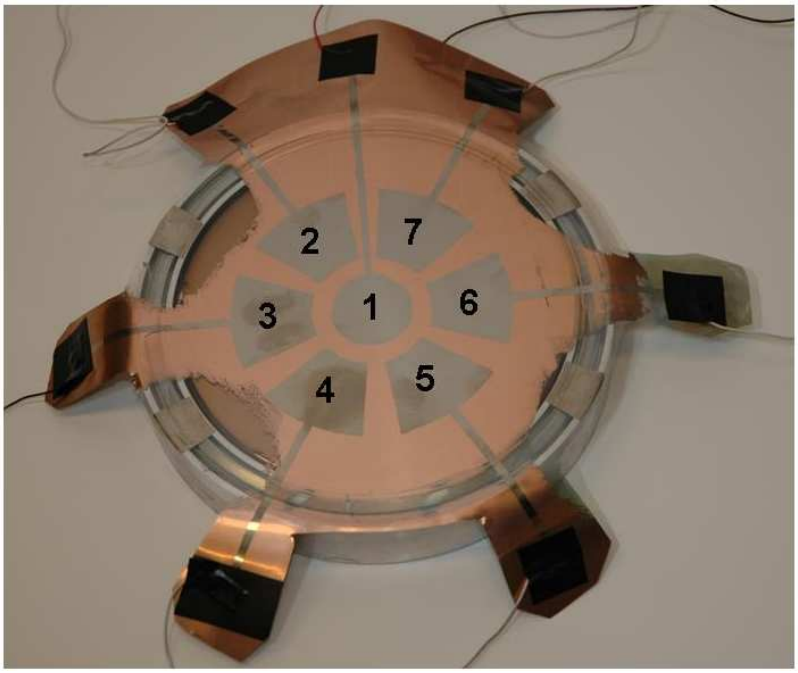

Figure 3: Actuator Pattern with Wires Attached to the Leads

In order to compare the test mirror to other optical structures, the areal density is calculated. Areal density is the total weight divided by the collecting aperture area. Equation 1 calculates the areal density, ρ_{areal}. Here m, SA, and r are the mass, surface area, and radius of the mirror.

$$\rho_{areal} = \frac{m}{SA} = \frac{m}{\pi r^2} \tag{1}$$

$$h = \frac{\rho_{areal}}{\rho} \tag{2}$$

In this case, the mass of the 5-inch diameter test mirror is 0.015 kg. The areal density of the membrane mirror is 1.184 $\frac{kg}{m^2}$. Equation 2 estimates the thickness of the silicone from the areal

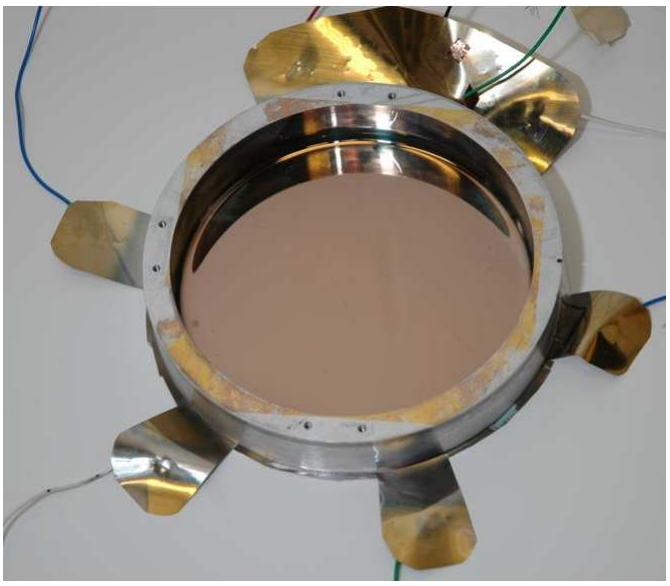

Figure 4: Reflective Side of Test Mirror

density. Here ρ, and h are the density and thickness of the silicone. The density of GE® Silicones RTV615 is known to be 990 $\frac{kg}{m^3}$, therefore the mirror thickness is 1.2 mm.

3.3.2 Modal Frequencies. The modal frequencies of the test mirror were measured for two reasons. First, modal frequencies are needed in order to set the data rate and to avoid exciting them during testing and control. Second, the modal frequencies are needed to calculate the in-plane tension of the test mirror. This calculation is needed to support a finite element model (FEM) of the test mirror developed by Shepherd [31]. The FEM will aid implementation of control algorithms and provide analysis towards improving the mirror.

Modal analysis was completed on the un-coated and coated membrane mirror using a scanning laser vibrometer manufactured by Polytec Inc. Best practices were used as described by Trad [35]. All tests used a set-up shown in Figure 5, with the mirror sitting right side up, and the test parameters listed in Table 2. The modal frequencies for the coated and un-coated mirror are listed in Table 3. As expected, the modal frequencies decreased in the coated mirror because the silicone adds weight without adding stiffness to cause dampening. Note that the un-coated modal frequencies are included for completeness and were not used to calculate the tension.

$$T = \rho \left(\frac{\omega_{mn} r}{\beta_{mn}} \right)^2 \tag{3}$$

Equation 3 was used to calculate the in-plane tension of the test mirror from circular membrane theory. Here, T is the tension per unit length, ρ is the areal density, r is the radius of the mirror,

β_{mn} are the roots of the Bessel function, and ω_{mn} are the corresponding natural frequencies in Hz. The values of β_{mn} and the tension calculated at each modal frequency are also listed in Table 3. The average tension of the test mirror is 18.585 $\frac{N}{m}$.

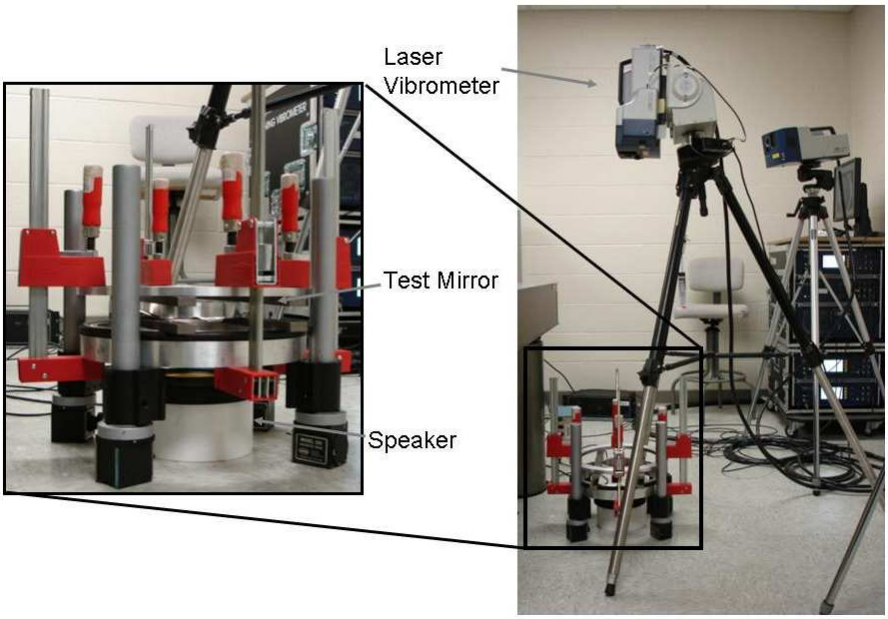

Figure 5: Laser Vibrometer Test Set-Up

Table 2: Table of Test Parameters

Parameter	Setting
Scan Points	529
Optimal Reading	99 - 100 %
Scanning Head	PSV-I-400 LR
Front End Software	PSV-E-400-3D (1D)
Acquisition Board	National Instruments PCI-4452
Acquisition Mode	FFT
Averaging	Complex
Average Count	10
Cosine Correction	Active
Bandwidth	100 Hz to 2 kHz
Sampling Lines	3200
Sampling Frequency	5.12 kHz
Sample Time	1.6 s
Resolution	625 mHz
Window	Rectangular
Overlap	50 percent
Actuation	60 V Periodic Chirp to Actuators

Table 3: Coated and Un-Coated Test Mirror Frequencies and Calculated In-Plane Tension

Mode	Un-coated	Coated	Tension $\frac{N}{m}$	βmn
1	353 Hz	146 Hz	17.597	2.4048
2	656 Hz	231 Hz	17.352	3.8317
3	696 Hz	246 Hz	19.678	3.8317
4	946 Hz	316 Hz	–	–
5	967 Hz	322 Hz	18.769	5.1355
6	–	353 Hz	19.523	5.5201
7	1244 Hz	398 Hz	18.591	6.3799

3.3.3 Leads. Manufacturing, storage, and testing of the mirror can cause breaks in the thin leads to each actuator on the PVDF material. These breaks either hinder or prevent the actuator from performing during testing. A method to test the leads was suggested by the manufacturer of the PVDF material (see Appendix A.4). To test the leads, the capacitance between the end of the lead and the mirror ground is measured. If the capacitance is less then 1.3 nf (or 1.31 $\frac{nf}{in^2}$) for any actuator, there is a break somewhere in the lead. The break is found by moving the positive node on the capacitance meter down the lead until the capacitance increases. To repair the lead, a thin layer of silver paint is painted over the break. Once it is dry, the capacitance is checked again.

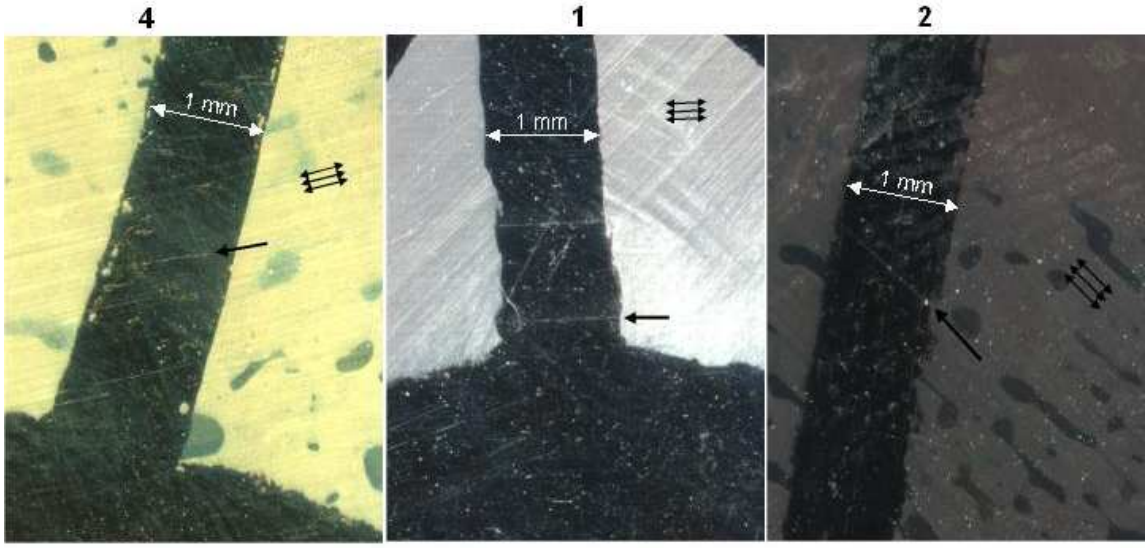

Figure 6: View of Breaks in Leads Under a Stereo Microscope

A Zeuss® stereo microscope was used to look at the lead breaks in a test mirror. Figure 6 shows three examples of the breaks. The single black arrow points to the break in the direction of the break, the three small black arrows indicate the x-direction of the PVDF material, and the number corresponds to which actuator the lead is for. The placement of the break was determined by using a capacitance meter. Notice that each break is in the x-direction of the PVDF material.

This was consistent for all leads except leads 3 and 6 which run parallel to the x-direction. Any break found in these leads ran along the edge of the frame that the PVDF material is glued to. This is believed to be caused from storing the mirror tightly between two glass plates and allowing the PVDF material to get creased between the mirror frame and the glass covering.

During testing there were some breaks that did not prevent the actuator from performing. It is not fully understood why this happens. It is expected that high voltage can arc across even very thin breaks. Also, until the first initial break in the lead is repaired, the capacitance meter test can not determine how many other breaks are further down the lead. Therefore, in this research, all leads are repaired by painting the entire length between the end of the lead and the initially measured break. A better way to determine all significant breaks within a lead needs to be found, as well as the effect (if any) of the repair using the silver paint.

3.4 Optical Test Set-Up

An optical test set-up with the WaveScope® WFS-01 Shack-Hartmann Wavefront Sensor (SHWS) manufactured by Adaptive Optics Associates (AOA) was used previously at AFIT and is documented by Sobers [32] [33]. The set-up is shown in Figure 7. Only a short description follows highlighting a few important features.

A 20 mW helium-neon laser (λ = 633 nm) was used to illuminate the test and reference surfaces via a beam splitter. The beam splitter separates the beam into two equal intensity beams. One beam is turned 90 degrees and reflected off a $\frac{\lambda}{20}$ flat mirror to return to the SHWS as a reference beam. The other beam is passed through the beam splitter, focused with a 1-inch doublet lens, and directed through a variable beam mask. The beam mask is adjusted for each test to illuminate only the desired area of the test mirror. This simplifies test subject area adjustment during calibration. The beam is reflected off the test mirror and returns to the SHWS as a test beam. The test mirror sits in a suspended horizontal position on the optics table, as shown in Figure 8, to allow the mirror to vibrate freely in the frame.

3.5 Data Acquisition System

A data acquisition system was developed to provide closed-loop feedback control to the test mirror. Data collection and closed-loop control of the test mirror surface is accomplished with the set-up illustrated in Figure 9. In summary, light enters the SHWS sensor head that measures micron level distortions in the test mirror surface and captures it in 42 Zernike coefficients[1]. These coefficients are exported through an ethernet cable to an Nport® ethernet-to-serial converter box.

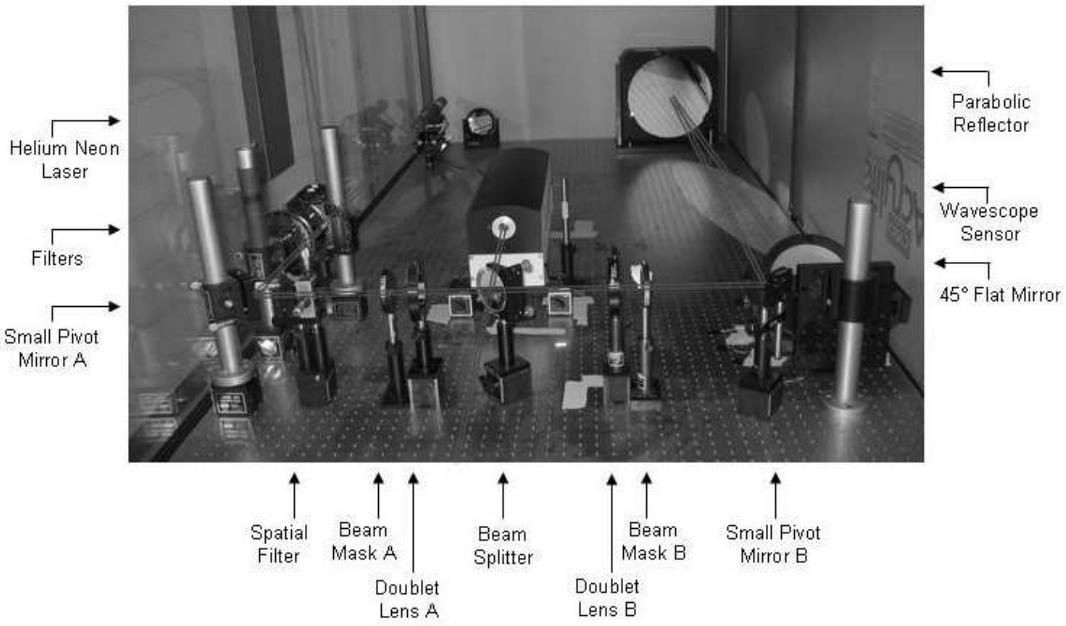

Figure 7: Optical Table Set-Up [35]

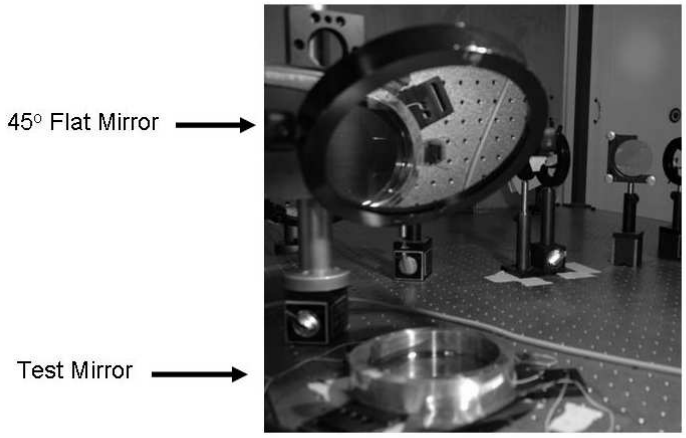

Figure 8: Test Mirror in Optical Set-Up

The serial cable connects to an interface board for dSpace®. dSpace® represents where the data conversion, data collection, and control implementation occurs. Actuator commands exported from the dSpace® signal generator go through a set of amplifiers before reaching the test mirror. A more detailed discussion of each part in the schematic follows.

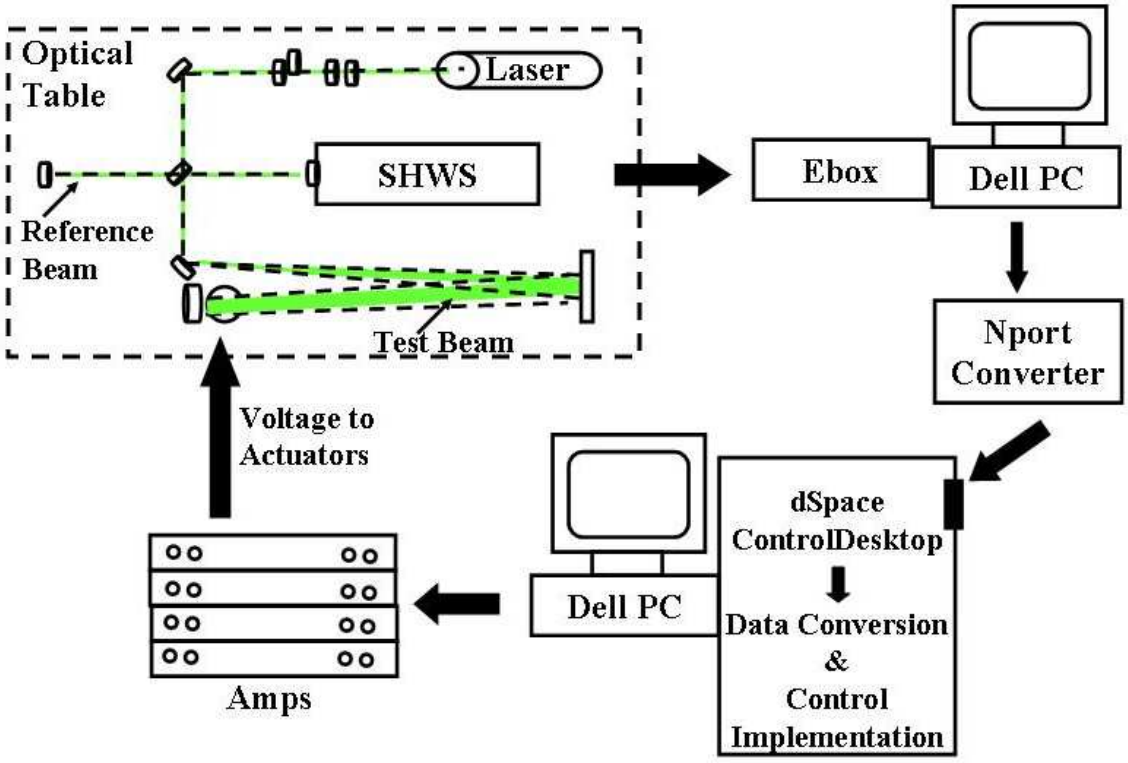

Figure 9: Schematic of Closed-Loop System

3.5.1 Shack-Hartmann WaveScope Sensor. All light entering the SHWS passes through a monolithic lenslet module (MLM) that focuses the light into a RS-170 monochrome CCD camera. The mirror surface measurement consists of the slopes of the wavefront in sub-apertures of the MLM. The fidelity of the data collected using the SHWS depends on the size and number of the lenslets in the MLM. Unless otherwise stated, a low resolution MLM with lenslets measuring 480 μm in diameter was used to capture a 3 in (0.0759 m) diameter area of the test mirror.

The SHWS acquisition software is loaded on the Dell® personal computer (PC) in TCL scripting language. The scripts configure all software functions such as GUI set-up, camera alignment and calibration, and Zernike coefficient calculations. During calibration, small regions of lenslets are

[1] A Zernike coefficient is the amplitude of a corresponding Zernike polynomial. An orthogonal set of Zernike polynomials makes up an optical basis function. A linear combination of them describes a surface. Forty-two coefficients is sufficient to describe the test mirror surface.

not bright enough to be measured. This is due to the scratches in the test mirror and unmodeled distortions in the optical path. The wavefront path difference of theses data points are fitted with a proprietary algorithm inside the scripts. For all recorded tests, the number of lenslets is recorded and ensured to be greater then 90 percent.

After calibration the live display mode in the software GUI is turned on for control tests. Normal operation calculates and displays Zernike coefficients. The software engineers at AOA wrote a script called *Socket.tcl* that enables the coefficients to be automatically exported across an ethernet cable at a rate up to 100 $\frac{megabits}{sec}$. Changing a few lines in the *Socket.tcl* and *TestEx.tcl* scripts switches the data format between ASCII characters and binary format (see Appendix A). The purpose of changing formats is to allow the data to be sent more efficiently. This is discussed in Section 3.7.5.

As shown in Figure 9, the SHWS hardware is interconnected to each other. The sensor head sends surface measurements to a video frame grabber in the PC via a RS-170 shielded coax cable. The PC sends commands to the Ebox via a 9-pin shielded serial cable. These commands tell the Ebox to move the CCD sensor during alignment and calibration. The Ebox is connected to the sensor head via a 25-pin shielded serial cable to receive commands and return status checks of the sensor.

The SHWS can reach measurement rates up to 4000 frames per second (FPS). Reaching this rate for control purposes is not feasible because as the FPS increase, the area and fidelity of the test mirror surface measurements decrease. One reason is that more light is required as exposure time is decreased. Another reason is that more data points require more computation time for Zernike coefficient calculations. A fast but efficient rate is needed to adequately measure and control the surface. The maximum rate that can capture a 3 in diameter area is 450 FPS. Therefore, the SHWS can sample the mirror surface fast enough to capture the first two modal frequencies.

3.5.2 Converter Box. The ethernet cable from the SHWS connects to a MOXA Nport® Express DE-211 ethernet-to-serial converter box. This conversion is required to allow dSpace® to acquire data through a serial port. The serial data stream is received by dSpace® through the RS232 serial port on the dSpace® hardware interface board. The converter box also sets the exported data stream characteristics through the MOXA software to match the RS232 serial port. For all tests, the data is sent at a baud rate of 115200 $\frac{bits}{sec}$ with 8 data bits, and 1 start bit.

3.5.3 dSpace. For this research, dSpace® imports data through an RS232 serial port on a PPC1103 dSpace® hardware interface board, relays it through dSpace® software, ControlDesktop, on a PC, and exports seven voltage signals through the signal generator.

The ControlDesktop software configures dSpace® through a data conversion model created in Simulink® and a layout created within the ControlDesktop GUI. Each model is loaded by dropping the model's SDF file into the active platform in ControlDesktop. Once loaded, any variable in the model can be displayed, plotted, automatically saved, and/or modified by digital gauges in the software. This gives the freedom to change any parameter during the control tests. Three models used in this research are discussed in Section 3.6. In general, each model reads in the Zernike coefficients and converts them to a string of decimal numbers. Directions on how to load a model are in Appendix A. The layout is set-up with three different types of gauges. A plotter array is set to capture the 42 Zernike coefficients. Slider gains are set to modify model variables and support real-time simulation. Digital displays were used to monitor the seven actuator commands.

The dSpace® signal generator exports seven actuator commands between plus and minus five volts. The test mirror is characterized to deform with voltages between plus and minus 600 volts. Therefore, each output from dSpace® is connected to a Trek PZD 700 Dual Channel amplifier to amplify the signal by two hundred. The output from each amplifier is verified using an Agilent 6.5 digital multi-meter. Voltage is received at the leads of each actuator on the membrane mirror directly from the amplifiers.

3.5.4 Post Processing. DSpace® is the critical factor in closing the loop for feedback control. Additionally, it is a means to efficiently record all test data. Each Zernike coefficient, actuator command, or any other model variable can be recorded for the entire test run time. ControlDesktop saves each desired variable as a structured array. This allows all post-processing to be completed within Matlab®. Movies of the surface and bar charts of the output signals can visually recreate the test. This makes data analysis easy to see. A complete list of script files used for post processing is in Appendix B.

3.6 Data Conversion

In order to implement a control scheme within dSpace® a data conversion model is needed to convert the incoming data. SHWS exports the 42 Zernike coefficients in either ASCII characters or binary format. Each format must be converted within dSpace® in order to create decimal number representation of the coefficients. This allows easy data recording and control implementation. Additionally, as decimal numbers, the coefficients can be multiplied by normalization factors to create an orthonormal set and have the mirror static bias subtracted from them. Three data acquisition models are described next. Each converts the data in three different ways. A summary of how each model was used in this research is included.

3.6.1 1225-Byte Data Conversion Model. The 1225-byte data conversion model is the Simulink® model, *Serial_1225.mdl* shown in Figure 10. This model is set-up to receive a byte-packet of 1225 bytes formatted in ASCII characters. It converts the entire byte-packet to a vector of 42 decimal numbers that are orthonormal Zernike coefficients. A procedure to load this model and a corresponding layout in ControlDesktop is given in Appendix A.1. A description of each block's function in the model is detailed below. Note that several blocks are common between the three models and will not be repeated in the other two descriptions.

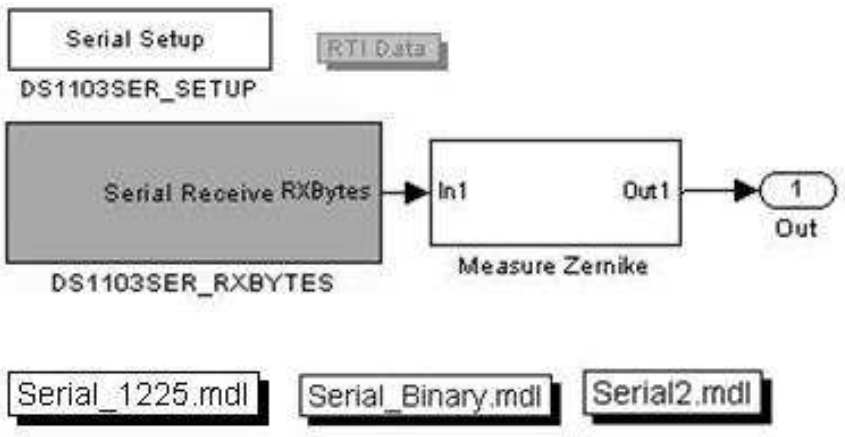

Figure 10: Data Conversion Model

In order to read to dSpace® input device, two dSpace® blocks are needed to define the serial port connection from the dSpace® interface board for each model. These are the $DS1103SER\_SETUP$ and $DS1103SER\_RXBYTES$ blocks. The block settings are set to maximum baud rate of 115200 $\frac{bits}{sec}$ with 8 data bits, and 1 start bit. In the 1225-byte model, they are also set to read 1225 bytes in a single time step. Each output byte from the $DS1103SER\_RXBYTES$ block is in 8-bit unsigned integer format.

The entire byte packet then moves to the $Measure\_Zernike$ block shown in Figure 10. For each model, the contents of this block change. The contents of this block for the 1225-byte model is shown in Figure 11. The byte-packet is converted to a double in the $uint8\_to\_double$ block before it reaches the $ASCII\_Converter$ block.

The function of the $ASCII\_Converter$ block is to convert each string of ASCII characters into 42 decimal numbers. It contains two c-coded $S\text{-}Function\_Builder$ Simulink® blocks shown in Figure 12. The $S\text{-}Function\_Builder2$ block converts the 1225 ASCII characters to 1225 integers.

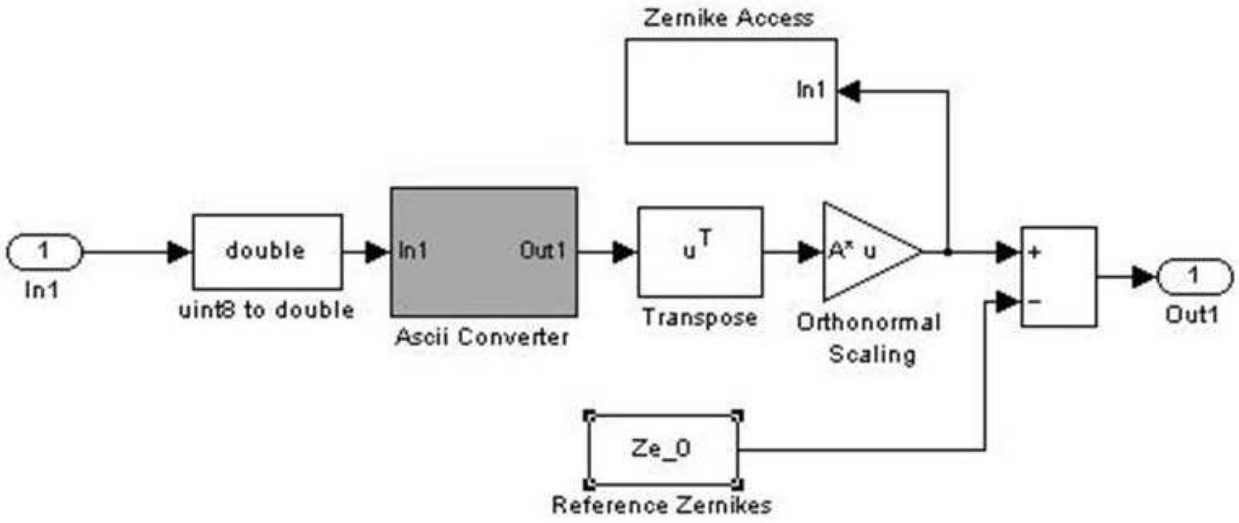

Figure 11: Measure Zernike Block in the 1225-Byte Data Conversion Model

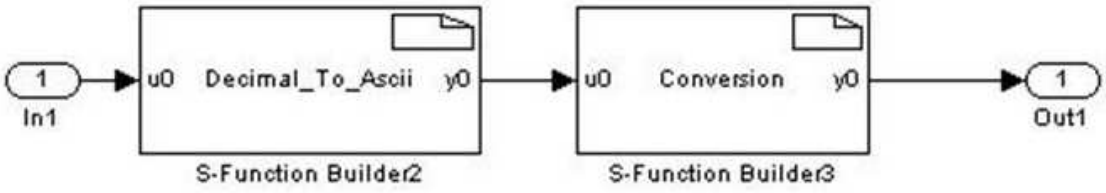

Figure 12: ASCII Converter Block of 1225-Byte Data Conversion Model

These integers can then be used to calculate a Zernike coefficient. The *S-Function_Builder*3 block calculates all 42 Zernike coefficients from the 1225 integers. The result is a vector of 42 coefficients in decimal format. The c-code scripts set in each block are given in Appendix C.1.

The vector of coefficients is transposed to dimensions 42 x 1 in the *transpose* block. This is necessary in order to multiply the vector by the *Orthonormal_Scaling* block containing a 42 x 42 matrix of normalization factors. The Zernike coefficients from the SHWS are orthogonal but not normalized to the unit circle. Therefore, normalization factors are needed to simplify surface deflection calculations by eliminating cross terms in the integration of Zernike polynomials. The normalization factors in *Orthonormal_Scaling* block matrix are discussed by Zhu [41] and recorded in Appendix D.2.

The *Zernike_Access* block is included to provide access to the normalized Zernike coefficients in ControlDesktop plotter array. It renumbers their numerical order so that ControlDesktop saves the coefficients in the same order as SHWS sends them. This is necessary because ControlDesktop does not order things by counting. An example of the reordering for the first ten Zernike coefficients is shown in Table 4.

Table 4: Zernike Access Renumbering Scheme

SHWS Zernike Order	1	2	3	4	5	6	7	8	9	10
Zernike Access Block Numbers	10	11	12	13	14	15	16	17	18	19

The *Reference_Zernikes* contains a vector of 42 numbers. They are the Zernike coefficients representing the test mirror surface at rest. This is a bias that can be subtracted because it is due to fabrication and mounting errors. This is initially set to zero and measured before all closed-loop control experiments begin. This makes the output of the *Measure_Zernike* block a set of orthonormal zernike coefficients that represent the test mirror's surface. This vector will be sent to a Simulink® control model that contains the control algorithm during closed-loop control tests.

3.6.2 1-Byte Data Conversion Model. The 1-byte data conversion model is the Simulink® model, *Serial*2.*mdl* shown in Figure 10. This model is set-up to receive one byte formatted in one ASCII character at a time. It combines each byte into a group of 29 bytes and converts each group into a single decimal number that represents an orthonormal Zernike coefficient. A procedure to load this model in ControlDesktop is given in Appendix A.2. A description of each block's function that is unique to this model is detailed below.

The $DS1103SER\_SETUP$ and $DS1103SER\_RXBYTES$ dSpace® specific blocks are set exactly as in the 1225-byte model except they are set to read one byte each time step. Each output byte from the $DS1103SER\_RXBYTES$ block is in 8-bit unsigned integer format.

Each byte then moves to the *Measure_Zernike* block. The contents of this block for the 1-byte model is shown in Figure 13. Each byte is converted to a double in the *uint8_to_double* block before it reaches the *Byte_Delay* block.

The *Byte_Delay* block sets up 29 model time-step delays. The purpose of this block is to recombine 29 bytes together that encode one Zernike coefficient. The result of this block is 29 data strings, each of which is one byte behind the one above it. Each is encoded in ASCII characters. This is a necessary step because there is no way to automatically put the right 29 bytes together. The next block will check for the string with the right 29 bytes and decode that one into 29 integers.

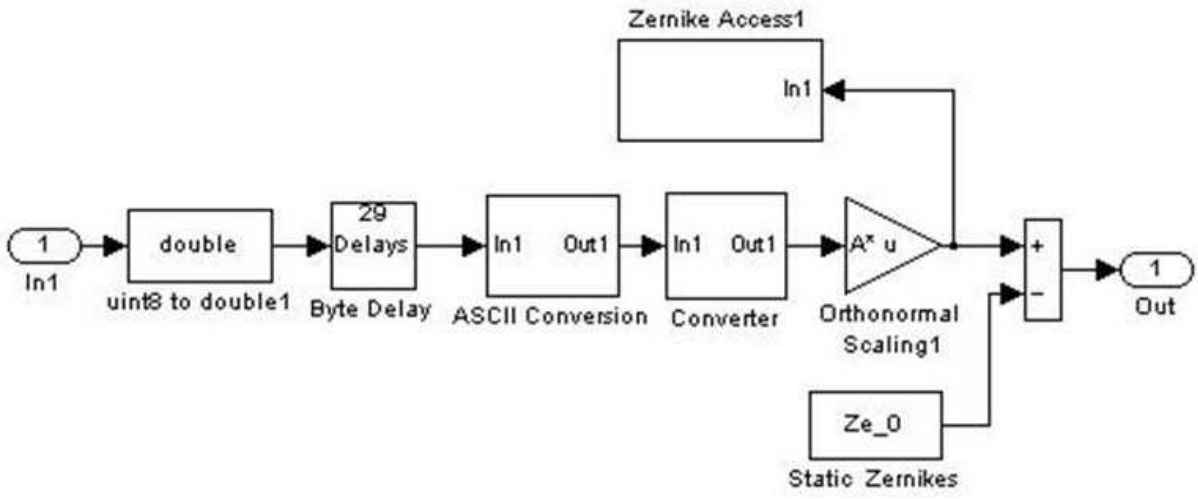

Figure 13: Measure Zernike Block in the 1-Byte Data Conversion Model

The *ASCII_Conversion* block filters the 29 ASCII character strings and converts the correct string to 29 integers. It contains two c-coded *S-Function_Builder* blocks shown in Figure 14. The *ASCII_Filter* block looks at each string of 29 bytes and does a comparison test to choose the right one. This comparison is based on finding the right position of the decimal point. Only one set of 29 bytes goes to the *Dec_to_ASCII* block. The *Dec_to_ASCII* converts the 29 ASCII characters to 29 integers. The c-code scripts set in each block are given in Appendix C.2.

The *Converter* block converts the 29 integers into a Zernike coefficient and places it in the correct place inside a 42 x 1 vector. It contains three c-coded S-Function Builder blocks shown in Figure 15. The *Zernike_Organizer* and the *Coeff_Number_Organizer* blocks calculate the Zernike coefficient and its corresponding mode number respectively from the 29 integers. The mode number is encoded in part of the 29-byte string. Both the Zernike coefficient and its mode number enter the *Zernike_Matrix_Builder* block. It uses the mode number to put the coefficient in the

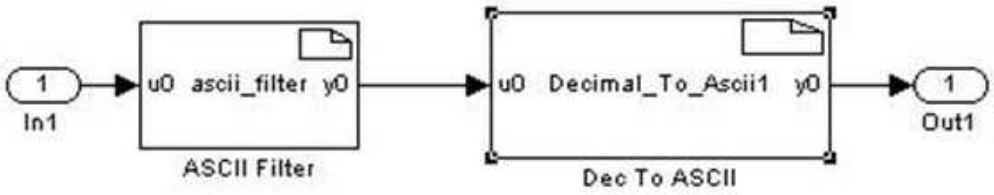

Figure 14: ASCII Conversion Block in 1-Byte Data Conversion Model

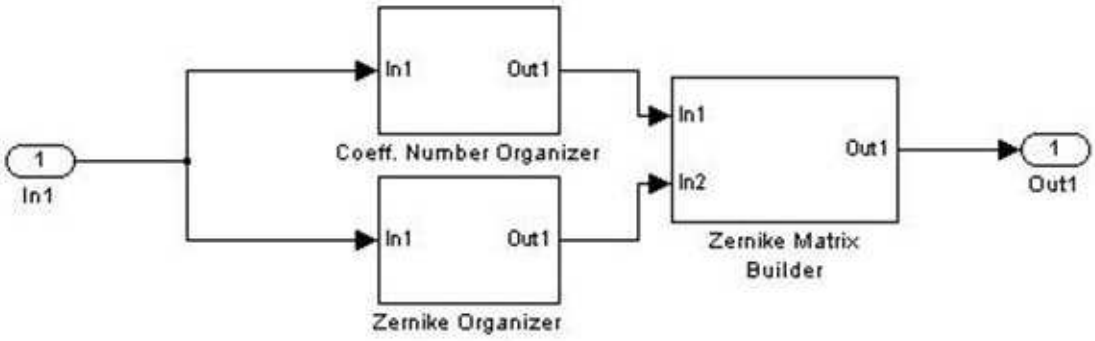

Figure 15: Converter Block in 1-Byte Data Conversion Model

correct space in a size 42 vector. Each coefficient value is held in the vector until it is replaced by a new coefficient. The result is that each coefficient populates a 42 x 1 vector that can be multiplied by the *Orthonormal_Scaling* block. The c-code scripts set in each block are in Appendix C.2. All other blocks are the same as described in the 1225-byte model.

3.6.3 8-Byte Data Conversion Model. The 8-byte data conversion model is the Simulink® model, *Serial_Binary.mdl* shown in Figure 10. This model is set-up to receive one byte formatted in eight zeros and ones, or binary format, at a time. It combines each byte into a group of 8 bytes and converts each group into a single decimal number that represents an orthonormal Zernike coefficient. A procedure to load this model in ControlDesktop is given in Appendix A.3. The procedure includes changing the SHWS TCL code to send the coefficients in binary format. A description of each block's function that is unique to this model is detailed below.

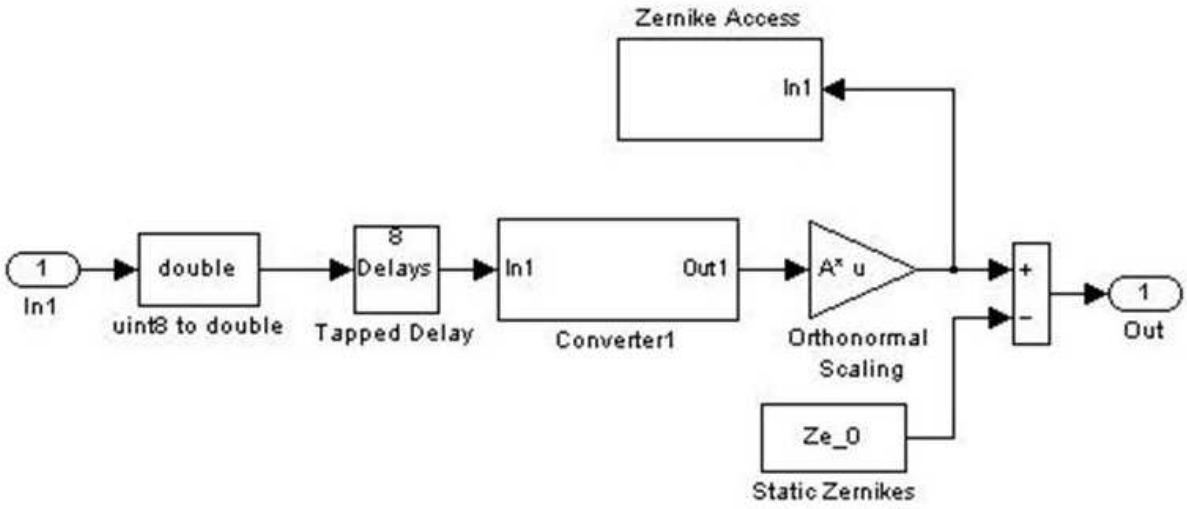

Figure 16: Measure Zernike Block in 8-Byte Data Conversion Model

The $DS1103SER\_SETUP$ and $DS1103SER\_RXBYTES$ dSpace® specific blocks are set exactly as in the 1225-byte model except they are set to read one byte each time step. Each byte cannot be read as individual bits. Therefore, each output byte from the $DS1103SER\_RXBYTES$ block is in 8-bit unsigned integer format that represents 8 zeros and ones.

Each byte then moves to the *Measure_Zernike* block. The contents of this block for the 8-byte model is shown in Figure 16. Each byte is converted to a double in the *uint8_to_double* block before it reaches the *Tapped_Delay* block.

The *Tapped_Delay* block sets up 8 model time-step delays. The purpose of this block is to recombine 8 bytes together that represent one Zernike coefficient and its corresponding mode number. The result of this block is 8 data strings, each of which is one byte behind the one above it. Each is encoded in an integer format. This is a necessary step because there is no way to automatically put the right 8 bytes together. The next block will check for the string with the right 8 bytes and use that one to calculate a Zernike coefficient.

The *Converter* block filters the 8 strings, converts the correct string a decimal number, and places it in the correct place inside a 42 x 1 vector. It contains two c-coded *S-Function_Builder* blocks and two Matlab® embedded blocks shown in Figure 17.

The *byte_Filter* block looks at each string of 8 bytes and does a comparison test to choose the right one. This comparison is based on finding the mode number. This number is an integer and can be identified by looking for two zeros in a row. Only one correct set of 8 bytes goes to the *Converting_MATLAB_Functions* blocks. The c-code scripts set in each block are given in Appendix C.3.

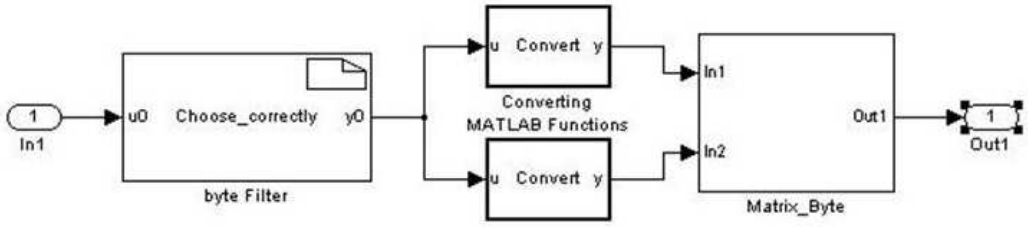

Figure 17: Converter Block in 8-Byte Data Conversion Model

The two *Converting_MATLAB_Functions* blocks simultaneous calculate the mode number and its corresponding Zernike coefficient from the 8 bytes. These are embedded MATLAB® m-files that could not be efficiently written because of the restrictions imposed from building the Simulink® model for real-time simulation. Additionally, each unsigned integer had to first be converted back to 8 zeros and ones and connected to three other converted bytes to create one long string containing 32 zeros and ones. This 32-bit string was determined to be in IEEE-754 Floating Point format. Since library Matlab® commands could not be used, 130 lines of code were needed to convert the string to a decimal number(see Appendix B.6). These computations restricted the maximum model time-step to 0.0006 seconds.

Both the Zernike coefficient and its mode number enter the $Matrix\_Byte$ block. It uses the mode number to put the coefficient in the correct space in a size 42 vector. Each coefficient value is held in the vector until it is replaced by a new coefficient. This block also eliminates erroneous data by checking for calculated coefficients exceeding a value of 10 and throwing them out. This additional step was added to the 8-byte model because huge numbers that were not real coefficient values would appear in the data most likely caused by a bit flip or a skipped bit. The bad data is replaced with the previous time-step value. The result is that each coefficient populates a 42 x 1 vector that can be multiplied by the $Orthonormal\_Scaling$ block. The c-code scripts set in each block are in Appendix C.3. All other blocks are the same as described in the 1225-byte model.

The main difference between each model is its performance during live simulation. The data conversion model effects the speed and amount of time that dSpace® can operate at. These two parameters are discussed in the next section. For this research, the 1225-byte model was used for open-loop and closed-loop control tests. The 1-byte model was used to capture the test mirror response characteristics in a linearization test. The 8-byte model was not used for any tests in this research but represents a step that could lead to a faster data acquisition system rates in the future.

3.7 System Data Rate

The goal of the new data acquisition set-up was to receive all 42 Zernike coefficients at a rate of 492 Hz from SHWS. There are two limiting factors on data rate acquisition. First, the maximum baud rate of the RS232 serial port connection is only 115200 $\frac{bits}{second}$. Second, each data conversion model has a minimum step-size that is restricted by the amount of computations in the model. An explanation of when each limitation affects the data rate and solutions to reduce these limitations are discussed below.

3.7.1 Data Format. There are currently two data formats that can be used to send Zernike coefficient values from SHWS. The first format is a 1225 byte-packet that encodes the 42 coefficients and their mode numbers in ASCII characters. Each byte-packet contains 29 bytes to encode each coefficient and their respective mode number, and 7 additional bytes at the end. The second format is a 338 byte-packet that encodes the 42 coefficients and their mode numbers in binary format. Each byte-packet has 8 bytes to encode each coefficient and their respective mode number and 2 additional bytes at the end. These two formats can be set in the TCL script language as outlined in Appendix A.

3.7.2 Byte-Packet Data Rate. There are two ways to receive either data format in a data conversion model. One way is to receive the entire byte-packet at one time and decode the entire packet into 42 decimal coefficients. This method is used by the 1225-byte model. In this case the data rate limitation is due to the maximum baud rate of the serial port. Equation 4 calculates the data rate.

$$DR = \frac{b}{gp} \tag{4}$$

Here DR is the data rate in Hz, b is the baud rate in $\frac{bits}{s}$, g is the number of bits in each byte, and p is size of the byte-packet. Table 5 lists the variables and calculated data rates for the ASCII and binary formats received in this way. Notice that the g is set to 10 $\frac{bits}{byte}$. This includes one stop bit and one start bit.

The system-determined DR is the measured rate that dSpace® could receive the data at. This rate is measured by first setting the SHWS to send the desired number of coefficients as fast as possible ($>$ 200 FPS for all tests). The 1225-byte data conversion model was set to record each coefficient at every time-step in ControlDesktop. Each time-step will reflect the previous coefficient until a new one is sent. This means that if every time-step changes, then the received data is being under-sampled. If the data does not change every time step, it is over-sampled and the number of points between each value change can be used to calculate the receiving DR. To calculate the determined DR, the number of data points it takes the first Zernike coefficient to change four times is counted and divided by four to get an average number of points between changing values. The determined DR is equal to the inverse of this average multiplied with the model time-step. Note that a data conversion model to capture a byte-packet in binary format was not achieved for this research and could not be measured.

Table 5: Byte-Packet Data Rates

Format	Number of Coefficients	Byte-Packet p	Baud Rate b	g	Calculated DR	Determined DR
ASCII	42	1225	115200	10	9.4 Hz	9.4 Hz
ASCII	36	1051	115200	10	10.96 Hz	10.98 Hz
ASCII	10	297	115200	10	38.8 Hz	34.5 Hz
ASCII	5	152	115200	10	75.8 Hz	34.5 Hz
Binary	42	338	115200	10	34.1 Hz	–

Efforts to increase the DR with this receive method included reducing the number of coefficients exported from SHWS because this reduces the size of the byte-packet. Results are shown in Table 5. Calculated DR for 10 and 36 Zernike coefficients based on Equation 4 closely matched the system-determined DR. They are not exactly the same because the determined DR is only an average over

4 time-steps. When only ten or five coefficients were exported, there is a limitation at 34.5 Hz that was not expected. The most suspect cause is the SHWS. Although it can measure the mirror surface at very high rates of speed, the export scripts may somehow be limiting the data transfer rate across the ethernet cable. A measurement of the data rate coming from the ethernet cord needs to be taken to ensure that SHWS is sending the data as fast as it is measuring the surface.

Additionally, another limitation to dSpace® is that there is a limited amount of memory space in the serial receive buffer. When the memory buffer is full, erroneous data is received. Receiving 1225 bytes 9.4 times a second only allowed data to be collected up to 80 seconds. To reduce buffer-size issues, a second way to receive data was developed.

3.7.3 One Byte Data Rate. The second way to receive data is one byte at a time and delay the simulation by either 29 or 8 time-steps for ASCII and binary formats respectively. The delay allows the model to lump enough bytes together to decode one coefficient into a decimal number at a time. In this case, the data rate limiting factor is the data conversion model time-step. Equation 5 calculates the expected data rate in this case. Here DR_t is the data rate in Hz, t is the model time step in seconds, d is the number of delays, and n is the number of coefficients being sent.

$$DR_t = \frac{1}{tdn} \tag{5}$$

The 1-byte ASCII and 8-byte Binary models receive data in this way. Table 6 lists the variables and data rates for the ASCII and binary formats. Because there are additional computations in the 8-byte model, its maximum model time-step is 0.0006s. Therefore, even though the number of delays and total bytes is less then the 1-byte ASCII model, the data rate is not faster with this receive method. However, since only one byte is received at a time, the memory buffer does not get full.

Table 6: 1-Byte Data Rates

Format	Number of Coefficients	Time Step t	Delays d	Calculated DR_t	Determined DR_t
ASCII	42	0.00009 s	29	9.1 Hz	9.1 Hz
Binary	42	0.0006 s	8	4.96 Hz	4.93 Hz

3.7.4 Lag. Although the memory buffer does not get full with the 1-byte receive method, during data rate tests, it was discovered that the 1-byte data conversion model can cause a time lag in capturing the data. This means that somehow the data is recorded on a slower time-scale than real time. This lag has only been noted to occur in one of the two situations. One, if ControlDesktop and SHWS are not set-up in the correct order initially or after a restart is done during testing. Two,

if testing continues for more than two hours. If a lag occurs, it easily observable by viewing the timing between ControlDesktop and SHWS software.

This lag was calculated for the 1-byte data conversion model and was confirmed experimentally. The lag factor is calculated by Equation 6. Here DR_t is the data rate in Hz and LF is the lag factor.

$$LF = \frac{1}{DR_t} \tag{6}$$

When the 1-byte model time step is set to 0.0001 seconds, the lag factor is 0.1218 by Equation 6. Therefore, the time scale recorded by ControlDesktop is divided by 1.1218 for an accurate time scaling during data processing. To confirm this experimentally, a low frequency sinusoid signal was sent to actuate the test mirror as in Figure 2. The first Zernike coefficient was plotted against the recorded time scale and a time scale with LF. The latter plot reproduced the sinusoid at the same frequency that is was sent. This LF was used to correct the time-scale captured during linearization tests (see Section 4.3.2).

3.7.5 Down-Sampling. A final parameter needed to set the data rate in ControlDesktop is down-sampling. It is used to match-up the system rate and the data capture rate in dSpace®. The down-sampling parameter sets the fraction of the data points to be saved. This is best illustrated with an example of the 1-byte data conversion model with a model time step of 0.001 seconds. The model time step sets the data capture rate to record 1000 data points every second. However, the system rate is only 8.2 Hz. By setting the down-sampling to 120, dSpace® records every 120^{th} data point, or about 8.3 data points a seconds. Setting this parameter saves space in the memory buffer during testing and in the structured arrays saved for post-processing.

As shown, the fastest data rate achieved by any of the models was less than 10 Hz unless fewer coefficients were exported. Forty-two coefficients are needed to sufficiently calculate surface deformation. Therefore, a maximum system rate of 9.4 Hz can be utilized in this research.

The two limiting parameters in the data conversion models with respect to the control tests is the system rate and time of operation. The 1225-byte model allows the fastest data rate at 9.4 Hz but fills the memory buffer within 80 seconds, or about 750 frames of data. This model was used to run all open-loop and closed-loop control tests but could not be used to characterize the mirror because of the operating time. The 1-byte model can achieve a system rate of 9.1 Hz and operate indefinitely. Therefore it was used for linearization tests described in the next chapter. During these tests, the time lag was checked and accounted for. Finally, the 8-byte model was not used in any test because it only achieved a system rate of 4.9 Hz. It was expected that faster rates could be

achieved with a binary format because the data is sent in much more efficient way. Simplifying the binary conversion process will make this possible. This is an area of further research.

3.8 Summary

This chapter described the complete data acquisition system needed for closed-loop control. The closed-loop system can capture the test mirror's surface deformation in 42 Zernike coefficients with a SHWS and output seven actuator commands through dSpace® signal generator at rates up to 9.4 Hz. This is not sufficient to capture or control the dynamics of the test mirror. The main limiting factor is the data conversion between SHWS and ControlDesktop. Three different methods to convert the data were described and lay the groundwork for future efforts to increase the system rate.

IV. Control Tests

4.1 Overview

This chapter describes the procedure and results of the open-loop and closed-loop control tests. The tests focus on the ability to generate four different surfaces via a measured relationship between the actuator commands and the surface deformation. The open-loop control tests demonstrate validity in the method used to capture the mirror's response characteristics and provide a basis of comparison for closed-loop tests. The closed-loop control tests characterize the feedback loop as a foundation to implement control designs on the test mirror.

The first part of this chapter defines the limitations of the test mirror and introduces a method to capture its response characteristics. The second part describes the control tests and presents the results.

4.2 Creating an Actuated Surface

A common method to demonstrate control in a deformable mirror is to create individual Zernike mode shapes or a combination of them. In general, greater spatial density of the mirror's actuators means a greater ability to produce any Zernike mode shape. The limitations of what surface shapes can be created depend on the number, shape, and distribution of the actuators. In this section, four different actuated surface shapes are created by the test mirror for the control tests.

The control algorithm used in this research (see Section 4.3) requires that the control surface shape be represented with 42 Zernike coefficients[1]. The vector containing the 42 Zernike coefficients is called $ZE_{control}$, where $ZE_{control} = [z_1, z_2, ...,z_{42}]$. $ZE_{control}$ needs to be achievable by the test mirror to demonstrate significant results. One way to define this surface is make the entire vector zero except for a single Zernike coefficient. The control surface will then be a single Zernike mode shape [18]. This method was not used because although the seven actuators can control a single Zernike coefficient, it was assumed that a surface created in the controllable subspace would show better results. A second method, used in this research, is described.

To capture an achievable $ZE_{control}$, a combination of actuator commands between -350 V and 350 V were applied to deform the test mirror. The 42 coefficients representing the resulting surface were recorded and set as $ZE_{control}$. Four combinations of actuator commands were chosen to be used for control tests. The actuator command combinations are listed in Table 7.

The results of these four combinations are shown in Figure 18. The view is of the reflective side of the mirror. The dashed lines represent the location of the actuators. The solid lines represent the

[1]This is only a limitation with respect to the data acquisition set-up. Since 42 Zernike coefficients are used to capture the test mirror surface, it can only be controlled by the same parameter. In general, some limitation will always exist with a wavefront measurement.

Table 7: Actuator Command Combinations to Create Actuated Surfaces

Actuator	One	Two	Three	Four
1	0 V	0 V	300 V	300 V
2	-300 V	245 V	-300 V	-250 V
3	300 V	350 V	-300 V	-250 V
4	-300 V	245 V	-300 V	300 V
5	300 V	-245 V	-300 V	-250 V
6	-300 V	-350 V	-300 V	-250 V
7	300 V	-245 V	-300 V	300 V

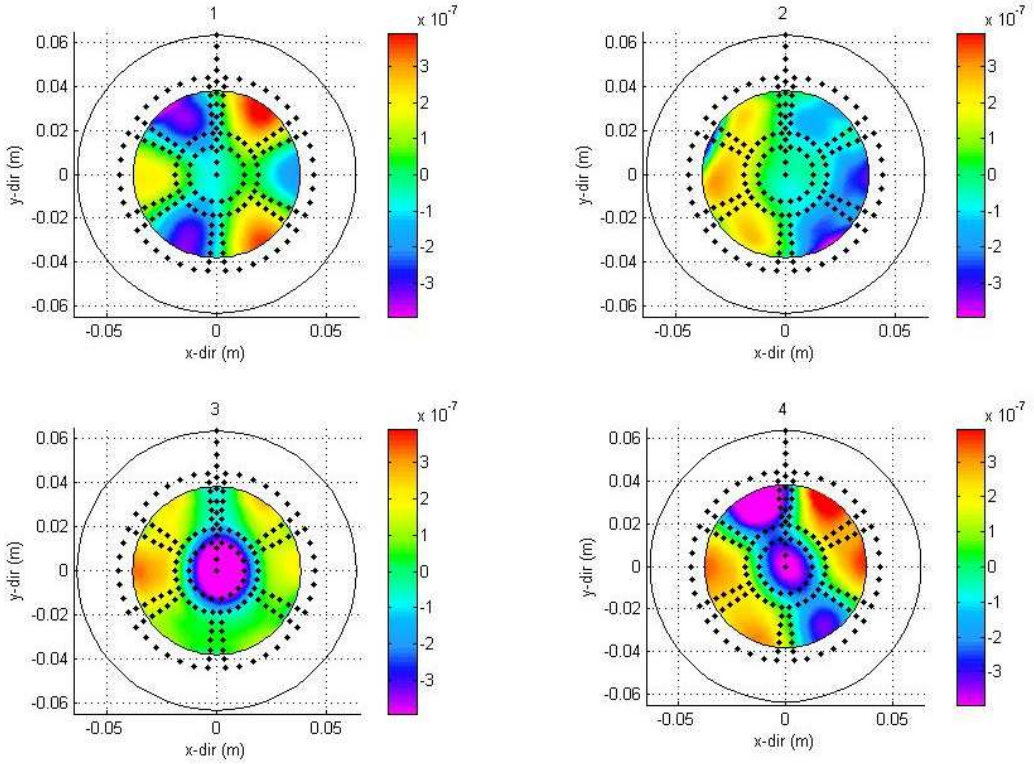

Figure 18: $ZE_{control}$ as Measured From Four Actuated Surface Shapes

total mirror surface and the 3 inch measured area. There is no measured surface data beyond the 3 inch diameter area. The shape is constructed from the Zernike polynomial expansion of $ZE_{control}$. The surface scale is given in meters. These surface plot characteristics are consistent throughout this chapter.

The actuator command combinations were chosen to create a recognizable pattern and remain within the actuation range of the test mirror. For example, the surface deformation of $ZE_{control}$ One shows actuators two, four, and six with positive surface deflection and actuators three, five, and seven with negative deflection. This combination creates an easily identified ripple effect around the mirror. Similarly, the surface deformations created by $ZE_{control}$ Two, Three, and Four can be identified as a tilt, donut, and taco shapes respectively.

A significant advantage for academic research to creating $ZE_{control}$ with this method is that the voltage signals needed to create the surface are known. They can be compared to the live actuator commands displayed in the ControlDesktop layout during testing. In the Controldesk software, an image of the test mirror surface cannot be displayed during testing and other test monitoring methods such as tracking all 42 Zernike coefficients is not reasonable. However, tracking seven actuator commands can provide a good indication of the surface deformation during simulation and provide immediate test validity. For all control tests, post-processing utilized the Zernike coefficient values and their corresponding surface deformation plots.

4.3 The Control Matrix

Now that an achievable surface shape has been determined for the test mirror, a method to control the mirror to that shape is needed. This is achieved by implementing an algorithm that utilizes a proportional controller that is based on the test mirror's response characteristics. The influence functions (IFS) capture the relationship between the mirror's surface deformation and the actuator commands. These IFS can be determined directly from surface measurements. In this section, a test to measure the IFS of the test mirror is described and the corresponding proportional controller, or control matrix, is calculated. The nonlinear response in the mirror is quantified to aid the assessment of applying a linear control method to the test mirror.

4.3.1 Influence Functions. A common way to implement a control scheme on a deformable mirror is to construct a constant proportional controller in a feedback loop. The proportional controller, or linear control matrix K, represents the response characteristics between the input and output. For this research, there are 42 inputs corresponding to 42 Zernike coefficients and 7 outputs

corresponding to 7 actuator commands. Therefore, K must have dimensions of 7 x 42. An extension of this discussion to other mirrors with additional actuators is intuitive.

Equation 7 is used to calculate K. Here K is simply the pseudo-inverse of H, the influence function matrix, where V is the actuator command vector, $V=[v_1,\ v_2,\ ...,v_7]^T$, and Z is a vector containing the Zernike coefficients, $Z=[z_1,\ z_2,\ ...,z_{42}]^T$. In this case H is also termed *modal control* because it is based on Zernike IFS rather than actuator IFS [10]. Zernike IFS are the ratios of the change in magnitude of each Zernike coefficient per one volt applied to the actuators.

$$HV = Z$$

$$K = (H^T H)^{-1} H^T \tag{7}$$

The influence functions are grouped in H by columns. The dimensions of H are 42 by 7, the number of Zernike coefficients and the number of actuators respectively. A linearization test is a method to calculate the IFS that make-up H and is described next. This method assumes a linear behavior within the actuator's range of operation (-600 to 600 volts).

4.3.2 Linearization Test. Seven Zernike IFS are needed to generate H. In this research, the IFS of the mirror are obtained by applying sequential 50-volt steps to one actuator at a time. Specifically, this linearization test was completed using the ASCII 1-byte data conversion model. No feedback was implemented for this test and the Zernike coefficients were captured from the $Zernike\_Access$ block.

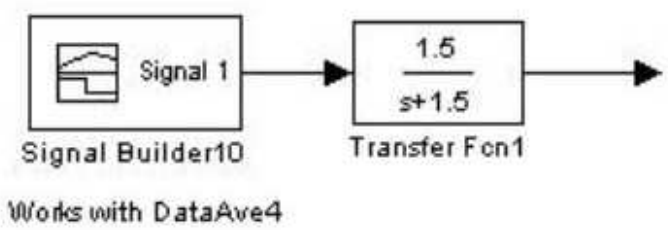

Figure 19: Linearization Test Actuator Signal Block with Low Pass Filter

The actuator signal in the linearization test is implemented with a Simulink® signal generator block. The actuator signals step up from zero to 600 volts, then down to -600 volts, and then back

up to zero volts. Each step is a 50-volt increment held for 20 seconds. Total test time for each actuator is 18.2 minutes. A low pass filter was initially applied to the signal to avoid exciting any unknown frequencies in the test mirror and was not improved once the mirror modal frequencies had been measured. This filter, shown in Figure 19, has a low cut-off frequency around half a Hz. Since the system rate is around 9 Hz, a more practical low-pass filter with a cutoff frequency around 4.5 Hz is recommended for further testing. This will decrease the time between each volt-step.

Post processing is completed with DataAve4.m (see Appendix B.1). The average Zernike coefficients at each step are used to calculate the difference in coefficient value between each signal step. The differences are averaged and divided by 50 volts to get the average change in each Zernike coefficient per commanded volt. Each set of 42 averages (one set per actuator) are arranged into the columns of the H, and K is calculated from Equation 7.

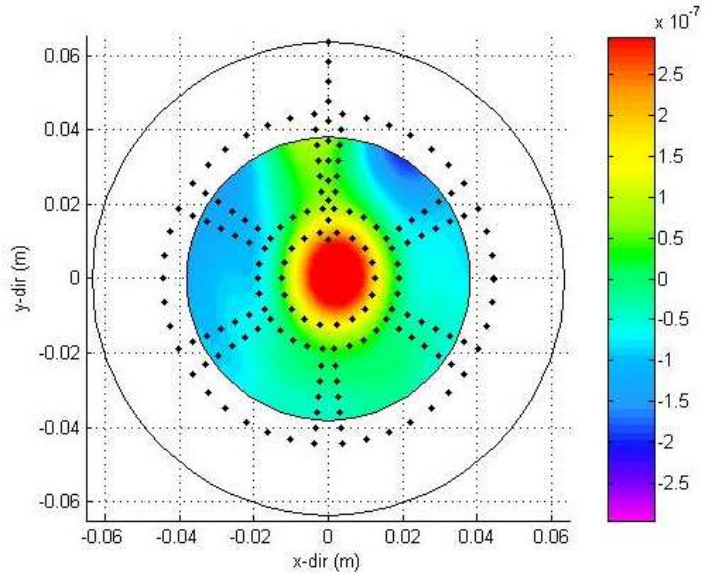

Figure 20: Influence Function of Actuator One

The IFS at 600 volts are plotted for each actuator in Figures 20 through 26. The shape is constructed from the Zernike polynomial expansion of the IFS. Note that since the IFS represent the average change per volt, the plots at -600 volts would have the exact opposite deformations plotted. The surface plots show the main deformation of each IFS remains within the region of its corresponding actuator but there is significant coupling between the actuators. Specifically, IFS four shows the actuator four region positively deformed as much as region five depressed. The opposite resulted in IFS five. A similar coupling response occurred between IFS two and five. Additionally, coupling in IFS one, three, and six caused significant depression in regions two and six, two, and one respectively.

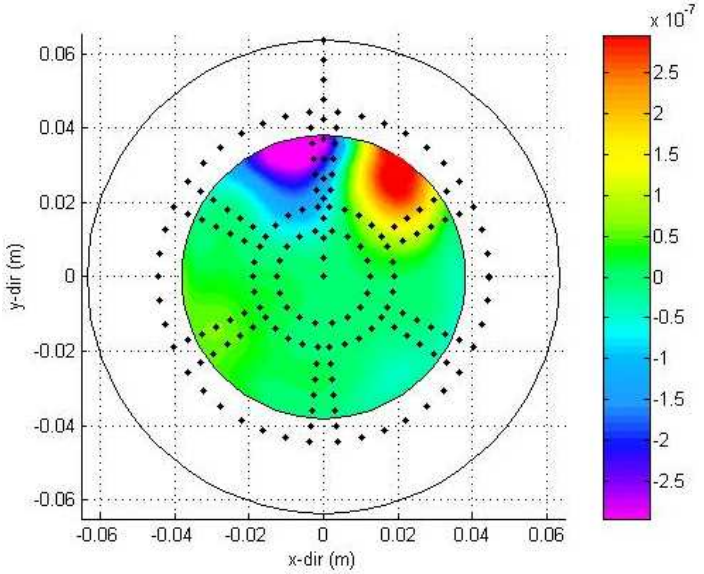

Figure 21: Influence Function of Actuator Two

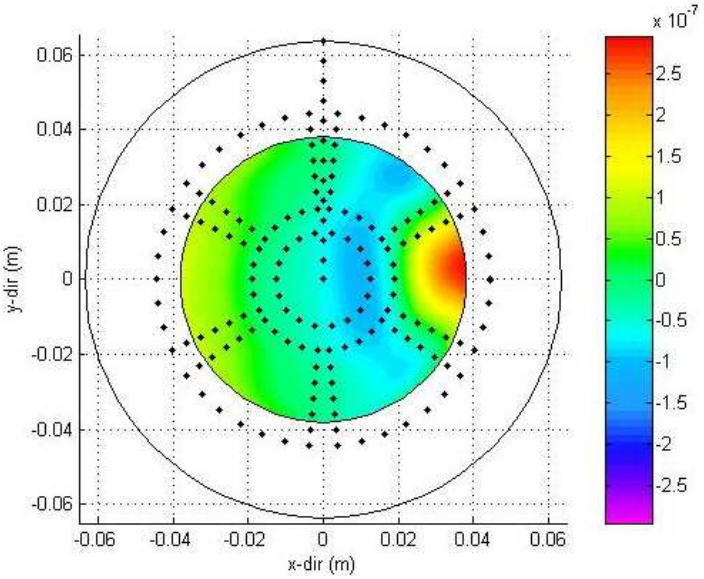

Figure 22: Influence Function of Actuator Three

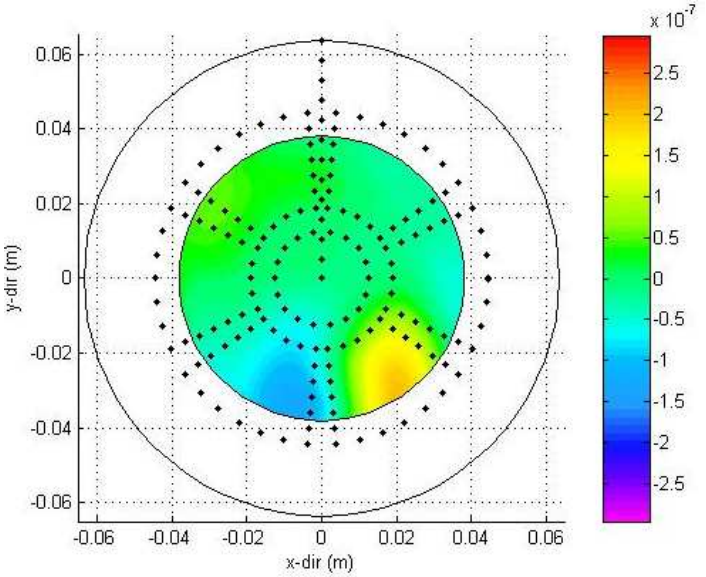

Figure 23: Influence Function of Actuator Four

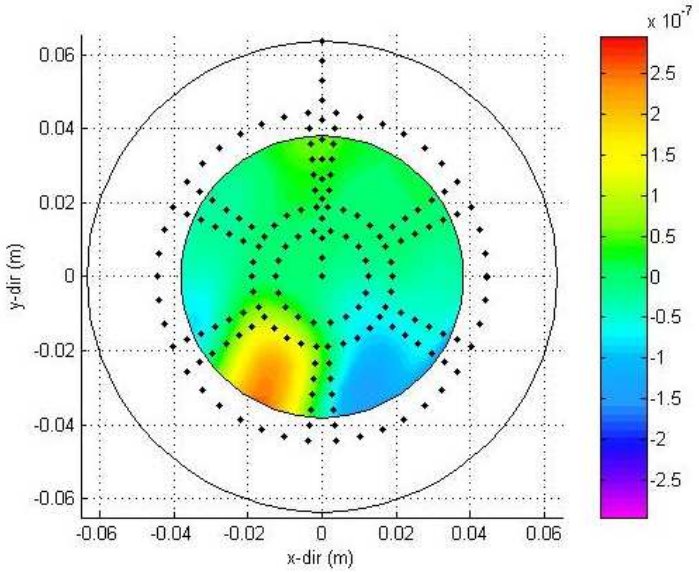

Figure 24: Influence Function of Actuator Five

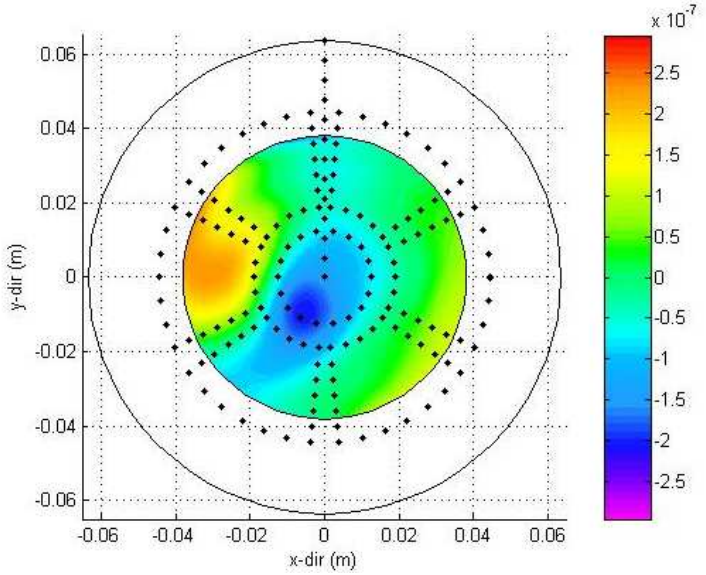

Figure 25: Influence Function of Actuator Six

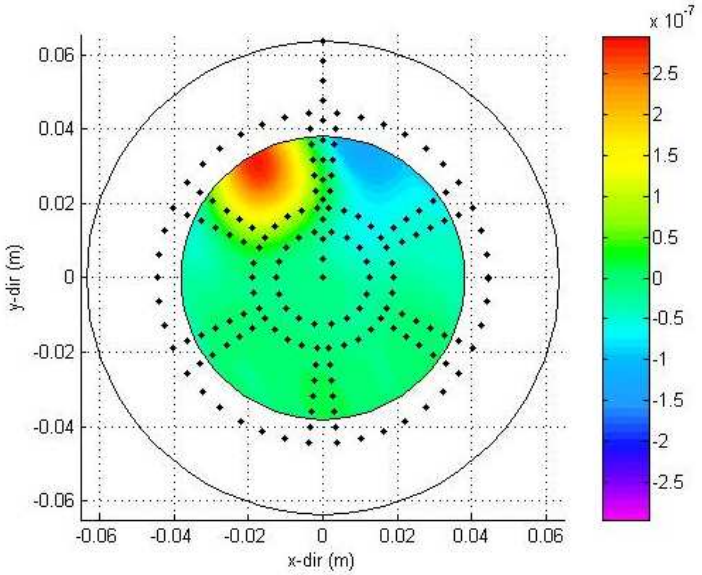

Figure 26: Influence Function of Actuator Seven

Notice that each IFS also does not have equal deformation capability between them. For example, the deformation caused by the IFS of actuator one is significantly greater than actuator five. The reason for this is twofold. First, the tension in the mirror is not even which prevents the response from being identical around the mirror. More significantly, the piezoelectric coefficient in the y-direction is over seven times the strength of the piezoelectric coefficient in the x-direction. The deformation of each actuator is significantly affected by its location with respect to the orientation of the PVDF material.

Another source of the difference between each IFS could be due to the construction and repair of the leads to each actuator. Manufacturing, storage, and testing of the mirror can cause breaks in the leads that can be repaired by painting silver paint over the breaks (see Appendix A.4). However, this can also change the actuator capacitance and deformation capability. Because of this, a measured IFS should always be used.

4.3.3 Calibration Curves. Quantifying nonlinearities in the test mirror deflection is necessary to assess the validity of calculating a linear control matrix. One way to observe them is with the linearization calibration curves. These curves are the total surface deflection plotted at each 50-volt step. The total surface deflection of the test mirror can be calculated from Equation 8.

$$D = (\sum_{i=1}^{42} z_i^2)^{\frac{1}{2}} \tag{8}$$

Here D is the total surface deflection in μm and z_i is each normalized Zernike coefficient in μm. This equation is a result of integrating the Zernike polynomials across the surface of the mirror.

Figures 27 through 33 show the total surface deflection at each voltage step in the linearization tests. The surface deflection in these plots is offset in order to put the negative zero-volt step at zero and dashed lines connect each 50-volt step to indicate whether the volt-step was positive or negative. The plots show that the surface deflection at the zero-volt step changes each time. This is a lag in the change of surface deflection, or hysteresis. For all actuators, the difference in this point did not change by more than 0.075 μm, suggesting in a hysteresis up to 10 percent. Since the volt-steps were held for 20 seconds, the lag shown in the calibration curves was expected to be much lower because adequate time was given for each actuator command to adjust. This suggests that calibration curves created with volt-steps nearing the control update rate of 9.4 Hz will show significantly more hysteresis. This will affect the amount of time it takes the control system to correct any error in the mirror surface.

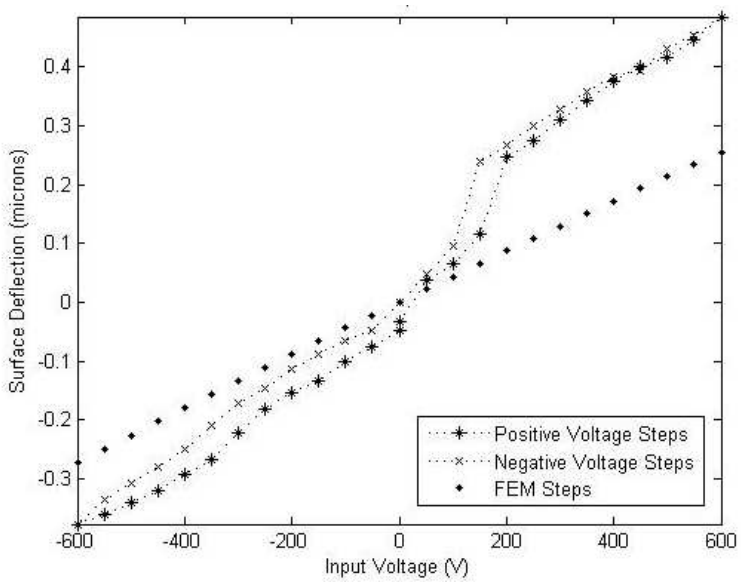

Figure 27: Total Surface Deflection for Actuator One over -600 v to 600 v

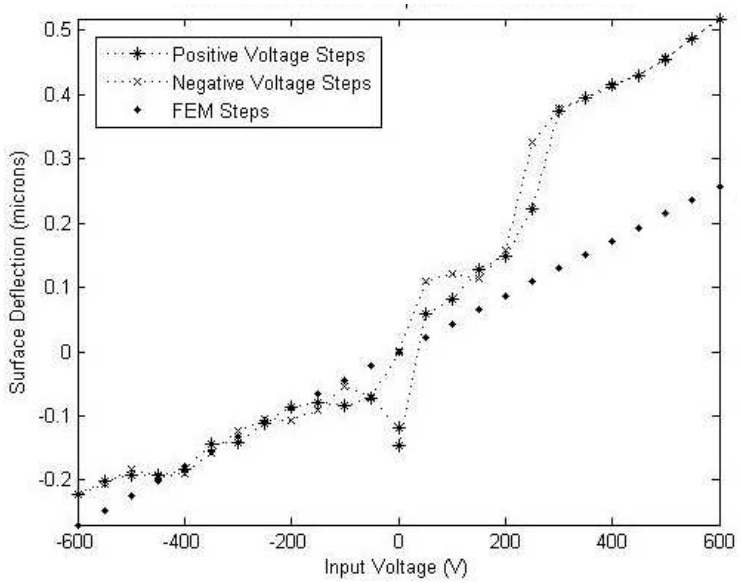

Figure 28: Total Surface Deflection for Actuator Two over -600 v to 600 v

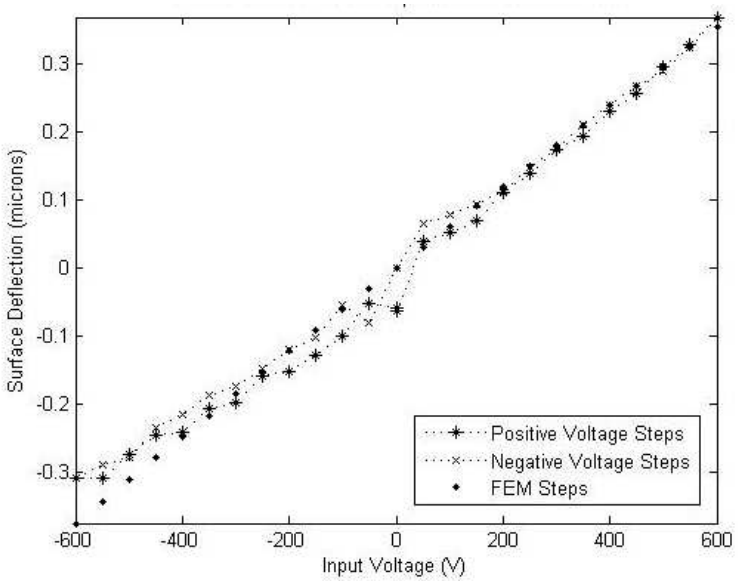

Figure 29: Total Surface Deflection for Actuator Three over -600 v to 600 v

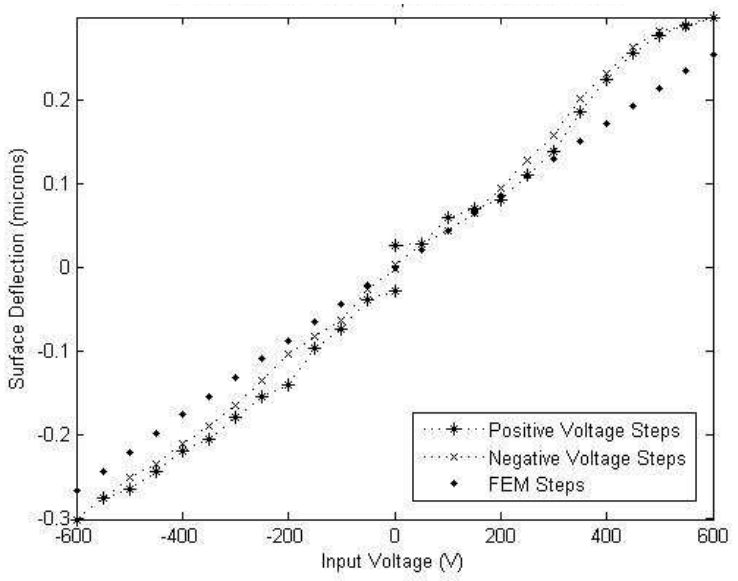

Figure 30: Total Surface Deflection for Actuator Four over -600 v to 600 v

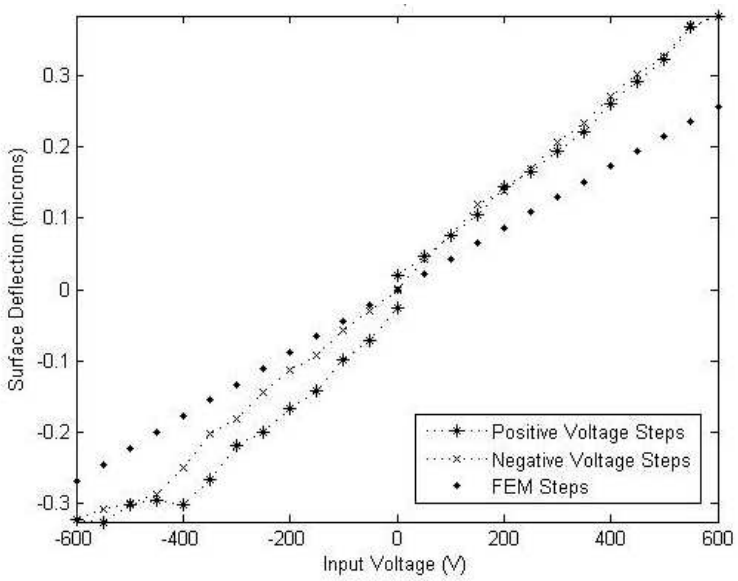

Figure 31: Total Surface Deflection for Actuator Five over -600 v to 600 v

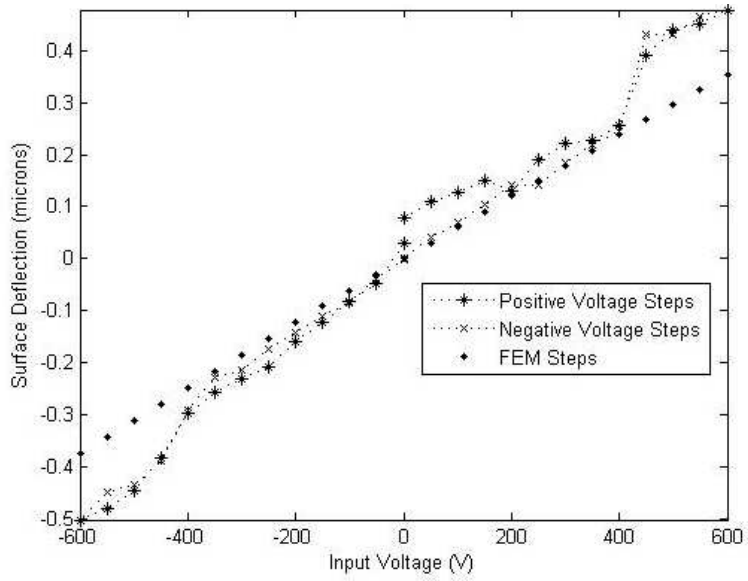

Figure 32: Total Surface Deflection for Actuator Six over -600 v to 600 v

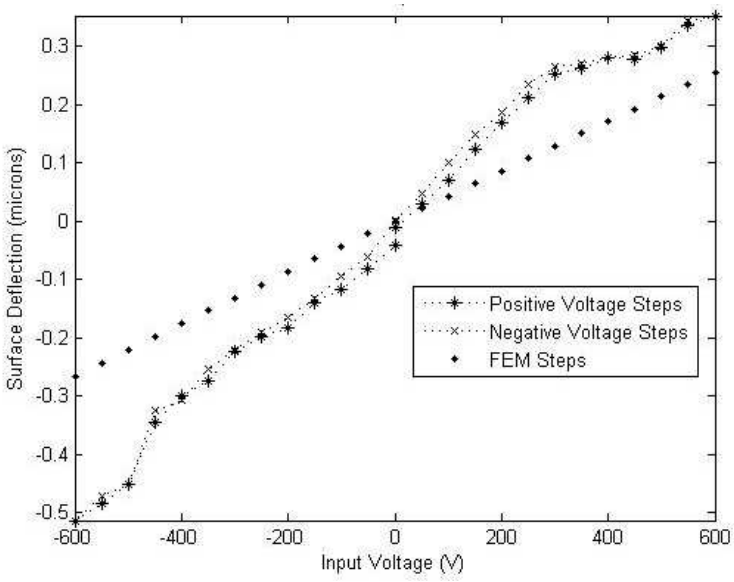

Figure 33: Total Surface Deflection for Actuator Seven over -600 v to 600 v

In addition to hysteresis, the plots show that the relationship between the actuator voltage and the surface deflection is not linear. Total surface deflections ranges between 0.56 and 0.92 μm. Notice that every actuator does not deflect the same amount for positive and negative actuator commands. This is emphasized by including the surface deflection predicted by a finite element model (FEM) of the test mirror [31]. The non-linear relationship allows the test mirror actuators to deflect up to 0.2 μm more than predicted in the FEM. Additionally, actuators one, six, and seven had a severe surface deflection change of more than 0.1 μm for a single voltage-step. It is possible that these three leads had an unexpected interaction with the silver paint used to fix them that enhanced the actuator deformation capability once enough voltage was available. In actuator two, there are two severe changes that significantly increase the positive voltage surface deflection. The second change around 200 V is consistent with the other leads, but it is not clear why there is a change around 0 V since this did not occur in any of the other actuators. A precise characterization of this phenomenon is recommended for future work in order to create a nonlinear block within the control scheme to remove these effects, or improve the mirror manufacturing to eliminate these effects. The nearly linear response observed in actuator three suggests that this is possible for this type of mirror.

4.4 Open-Loop Control Test

The control tests are implemented by rearranging Equation 7 and setting Z to $ZE_{control}$ as shown in Equation 9. There are two main reasons why the actuator commands, V, calculated in this equation will not create the four $ZE_{control}$ surfaces exactly. First, K is created by a method that

assumes the test mirror is linear. As shown in the calibration curves, there is not a linear relation between the actuator commands and the resulting surface deformation across the entire operational range. Second, the mirror's nonlinearities are included in the computation of H, and hence K, and may cause V to exceed the operational range of the actuators. These commands will saturate the model and be automatically set to the extreme voltages (-600 V and 600 V). For this reason, it is important to determine what actuated surface is expected to be generated by Equation 9.

$$K\ ZE_{control} = V \tag{9}$$

In the next section, the actuated surfaces represented by $ZE_{control}$ are created through open-loop control. The open-loop test results demonstrate whether K captured the mirror response characteristics well enough and provide a basis of comparison for closed-loop tests. The variation in the surface measurement during open-loop tests is compared to the variation in the surface measurement of a mirror at rest. This is included to determine if actuating the mirror causes a change in the magnitude and frequency of the mirror's natural fluctuation.

4.4.1 Set-Up. The open-loop tests were implemented using the data acquisition system with the ASCII 1225-byte data conversion model described in Chapter Three. This set-up provides the data acquisition rate of 9.4 Hz. All tests run for 30 seconds which captures the actuated surface 282 times. For all tests, a time-step vector and Z are recorded in a structured array and all post-processing is done with the script $ZernikeData.m$ and $ControlTest.m$ (see Appendix B.3 and B.9).

Additionally, each actuator is commanded within ControlDesktop via the control model shown in Figure 34. The seven DAC blocks are dSpace® coded blocks that provide analog signal to the dSpace® signal generator. These seven blocks correspond to the seven actuator commands. The subfunction shown between each DAC block and the control matrix is shown in Figure 35. Each DAC block is hard-coded to amplify the input signal by a magnitude of ten. The preceding $Cancel\_Dac$ block accounts for that magnitude by dividing it out. A saturation block is included to prevent the actuator commands from exceeding the operational range. These seven blocks are used during all control tests.

4.4.2 Results. To determine the expected actuated surfaces, open-loop actuator commands were calculated by Equation 9 with each of the four $ZE_{control}$ and implemented as shown in Figure 34. The average of 188 Z vectors is set as $ZE_{expected}$. Figure 36 shows the surface plots of $ZE_{expected}$ and Table 8 lists the actuator commands calculated by Equation 9 for each test. Additionally, the standard deviations of $ZE_{expected}$ are 0.0076 μm, 0.0092 μm, 0.0045, and 0.0043 μm respectively for

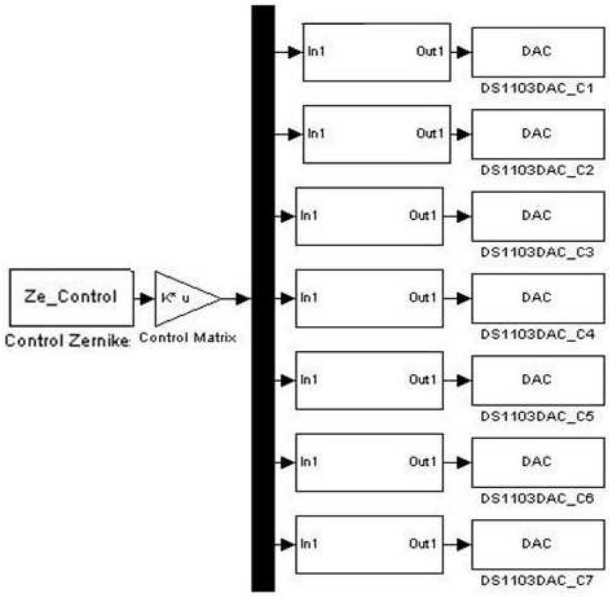

Figure 34: Control Model Set-Up for Open-Loop Tests

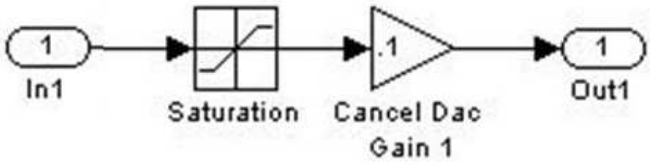

Figure 35: Inside View of Subfunction Leading to DAC

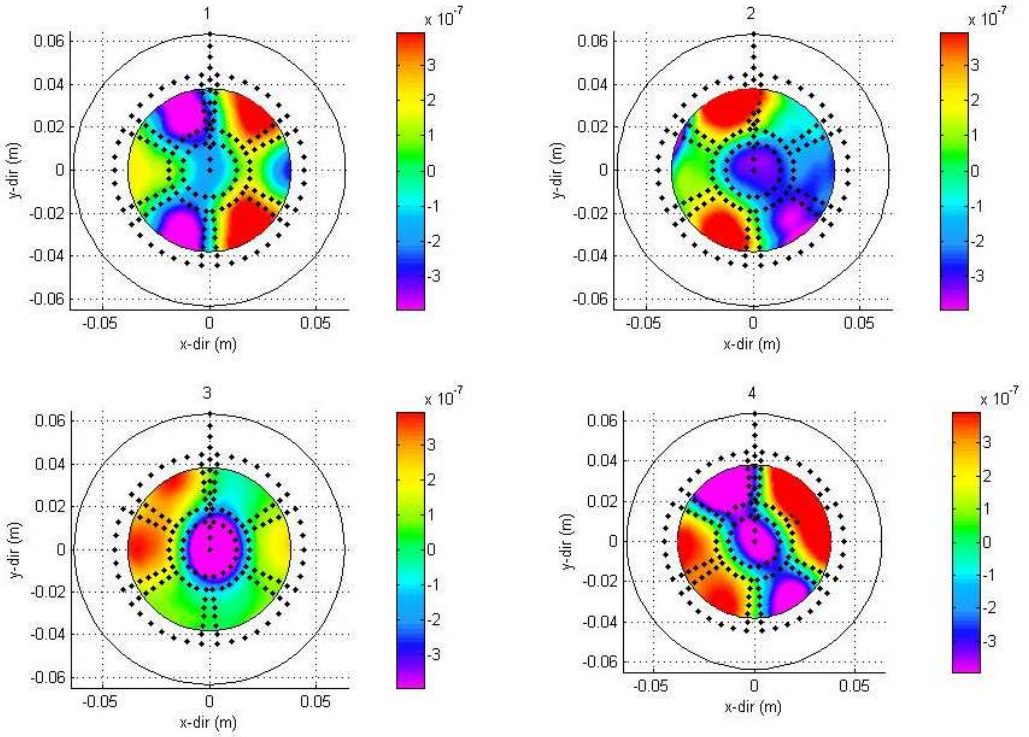

Figure 36: $ZE_{expected}$ as Measured These Four Actuated Surface Shapes

each test. These four surfaces are used for comparison to closed-loop feedback results in the next section.

A comparison between the original and calculated actuator commands and their corresponding surface shapes is a measure of how well the linear control matrix captured the nonlinear response of the test mirror. In Figure 36, the four shapes are still recognizable as a ripple effect, a tilt, a donut, and a taco. However, the residual error between $ZE_{expected}$ and $ZE_{control}$ is significant. The residual error, R_e, is calculated by Equation 10 as the root mean square between $ZE_{control}$ and $ZE_{expected}$.

$$R_e = [\sum_{i=1}^{42}(ZE_{controli} - ZE_{expectedi})^2]^{\frac{1}{2}} \tag{10}$$

For each test, R_e was calculated to be 0.3318 μm, 0.5 μm, 0.24 μm, and 0.38 μm respectively. The residual error shows on average a 50 percent increase in the total surface deflection of the test mirror. This is also reflected in the average difference of 225 volts between the calculated V listed in Table 8 and the original V. The change in V results in three of the four tests to have a calculated V that exceeds the operational range of the actuators. The significant change in actuator commands is not explained by the hysteresis in the test mirror since it was calculated to be about 10 percent.

Additionally, these changes are more significant then can be accounted for by the changes to the test environment during the duration of these tests. It is most likely that the nonlinearity of the mirror needs to be captured in multiple ways. This could be explored by changing the volt-step pattern, magnitude, and duration during the linearization test. This could reveal, for example, that there is a significant difference in the relationship between the actuator response to two 50 volt steps verses a single 100 volt step. Identifying the nonlinear effects more precisely would help to incorporate them into the controller.

A similar comparison was made between the original V and those calculated with Equation 9 using the control matrix determined from the FEM. It was expected that the difference between the actuator commands would be significantly less. On average, the calculated V was about half the difference, or 125 volts different then the original V (see Table 9). This is a significant improvement however further efforts to center the mirror in the test set-up and adding hysteresis to the model should improve this.

Table 8: Actuator Command Combinations Calculated by Equation 9

Actuator	1	2	3	4
1	258 V	56 V	851 V	491 V
2	-141 V	-50 V	-5 V	-462 V
3	613 V	160 V	-155 V	-402 V
4	-372 V	408 V	-126 V	737 V
5	540 V	-526 V	-154 V	-366 V
6	-162 V	-279 V	-242 V	-315 V
7	829 V	-581 V	-158 V	953 V

Table 9: Actuator Commands Calculated with the FEM Control Matrix

Actuator	1	2	3	4
1	86 V	-65 V	587 V	294 V
2	-404 V	300 V	-201 V	-587 V
3	224 V	236 V	-367 V	-406 V
4	-398 V	243 V	-358 V	396 V
5	540 V	-145 V	-271 V	-711 V
6	-160 V	-201 V	-225 V	-404 V
7	434 V	-278 V	-158 V	268 V

4.4.3 Surface Variation. The test mirror surface deflection constantly fluctuates due to air currents, measurement noise, and other structural vibrations. This wobble was indicated in the open-loop test results by the standard deviation of each surface plot which is an average of 0.0076 μm. One intention of closed-loop control is to maintain or reduce the magnitude of these fluctuation. Therefore, the magnitude of this fluctuation is significant for comparison to closed-loop feedback control surfaces. Additionally, if the measured variation is significantly different in an actuated surface, this is an indication that the mirror will not perform well in closed-loop tests. To

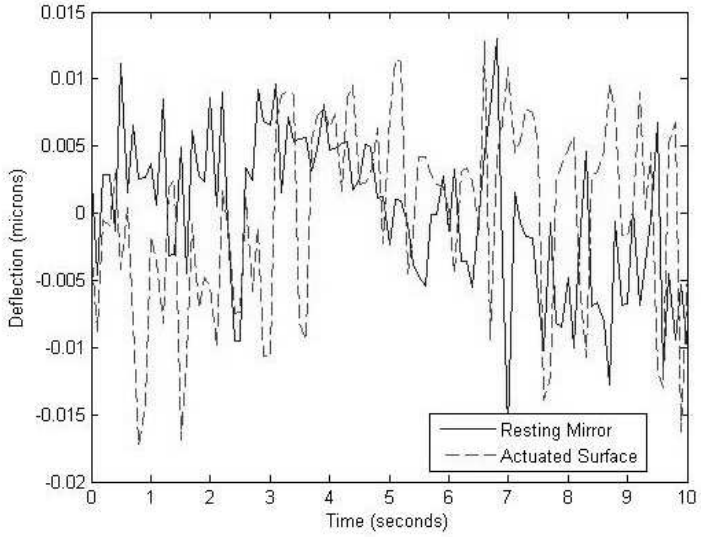

Figure 37: Time History of Controlled Surface

quantify it, the test mirror surface was recorded with no actuation and compared to the four actuated surfaces above. The time history of the total surface deflection per Equation 8 for the resting mirror and actuated surface Two is shown in Figure 37. The change in deflection magnitude of the test mirror prior to actuation was similar with all four of the actuated surface tests. For all tests, the change in deflection magnitude was less than 6 percent of the total commanded deflection. This means that actuating the mirror with constant actuator commands does not significantly increase the fluctuation in the surface of the mirror. In general, it remains around 6 percent during static deformation testing.

4.5 Closed-Loop Control Test

In general, an open-loop control system can not be designed to react to changes in the surface or actuators of the mirror that may be caused in the space environment. With the right design, closed-loop control can react to changes and correct them. Demonstrating a simple feedback control system is the first step to design and implement more advanced control methods. In this section, a closed-loop feedback system is implemented with the control matrix to validate the data acquisition system set-up and verify the mirror's performance capability. The feedback control design presented is a variation of an iterative control method presented by Fernandez [18]. It uses a convergence rate that can be adjusted to compromise steady-state error for stability. Although steady-state error is not desired, this method was chosen to ensure a fast convergence rate was achievable with stability

and simplicity. This design is described and the results of four tests, one for each $ZE_{control}$, are given.

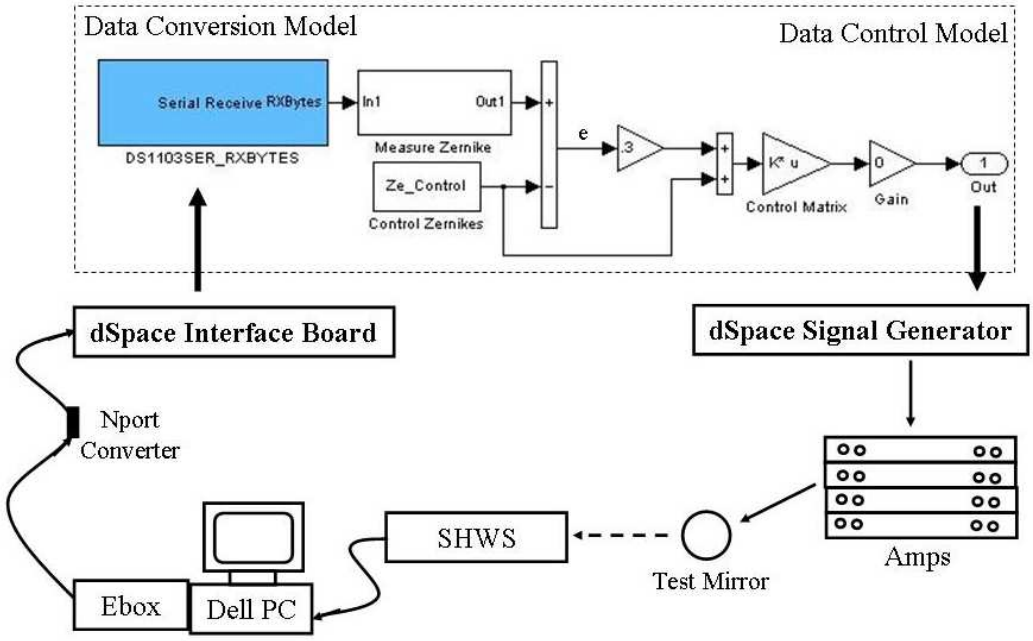

Figure 38: Schematic of Closed-Loop System

4.5.1 Set-Up. The schematic shown in Figure 38 illustrates the control scheme along with the entire closed-loop system. It shows the connection between the ASCII 1225-byte data conversion model and the dSpace® signal generator that sends the analogue commands to actuate the test mirror. This control scheme is also captured by Equation 11.

$$e = Y_{(k-1)} - ZE_{control}$$

$$V_k = 0.3\ K\ e + K\ ZE_{control} = K(0.3\ Y_{(k-1)} + 0.7\ ZE_{control}) \tag{11}$$

Here, e is the error between $ZE_{control}$ and $Y_{(k-1)}$, where $Y_{(k-1)}$ is the measured Zernike coefficients out of the $Measure\_Zernike$ block. The iteration count is k. The error is multiplied by a factor of 0.3 and added to $ZE_{control}$. The calculated actuator command vector, V_k=[v_1, v_2, ...,v_7], is then sent to the seven DAC blocks connected to the signal generator.

This control scheme was originally based on the iterative control method presented by Fernandez for a 37 actuator deformable membrane mirror [18]. It is represented by Equation 12. This

method requires that the previous actuator command, $V_{(k-1)}$, is known. To implement this method in a control model, a delay block is needed to hold the previous actuator command. In order to simplify the control model, the method was adjusted by replacing $V_{(k-1)}$ with the initial actuator command equal to $K \times ZE_{Control}$.

$$V_k = 0.35\ K\ e + \ V_{(k-1)} \tag{12}$$

The expectations for this control scheme are drawn from the results presented by Fernandez. First, it is expected that the variation in the control method will not significantly affect the stability or convergence rate properties of the original scheme. Although these properties are inherent in the 0.3 factor and therefore indirectly included in the calculation of $V_{(k-1)}$, it is expected that the main contributor is from the first half of the equation. This means that the residual is expected to reach the steady-state value within 10 iterations or 1.06 seconds with no significant overshoot.

Second, since the $V_{(k-1)}$ is replaced with a constant, the steady-state errors are expected to be larger than the 3 to 8 percent reported by Fernandez. Additionally, the maximum amount of steady-state error expected is a third of the total surface deflection of $ZE_{expected}$. This was determined by examining the open-loop part of Equation 11. By setting $Y_{(k-1)}$ to zero, V_k will be commanded to a seventh of $K \times ZE_{control}$. This will command a surface deformation equal to 30 percent less than $ZE_{expected}$.

The layout in ControlDesktop is set-up to capture each time-step of Z with a plotter-array, display the seven actuator commands in digital display gauges, and provide real-time changes to the *Gain* block magnitude with a slider-block. For each test, the *Gain* block is initially set to zero so that no feedback signals reach the signal generator. This gain is switched to one during each test to close the feedback loop. This allows ControlDesktop to capture Z before and after feedback starts. For all tests, ControlDesktop is set to record at least 50 seconds of data and feedback is started within 10 seconds after data recording starts. The closed-loop tests were implemented using the data acquisition system with the ASCII 1225-byte data conversion model shown in Figure 10. The data acquisition rate for all tests is 9.4 Hz. For all tests, a time-step vector and Z are recorded in a structured array and all post-processing is done with the script $ZernikeData.m$ and $ControlTest.m$ (see Appendix B.3 and B.9).

4.5.2 Results. The results of four quasi-static closed-loop control tests are reported in this section. Each test number corresponds to one of the four $ZE_{control}$ numbers.

The average surface obtained during each test and the average residual between the obtained surface and the expected surface are plotted in Figures 39 to 46. These plots are averaged over 180

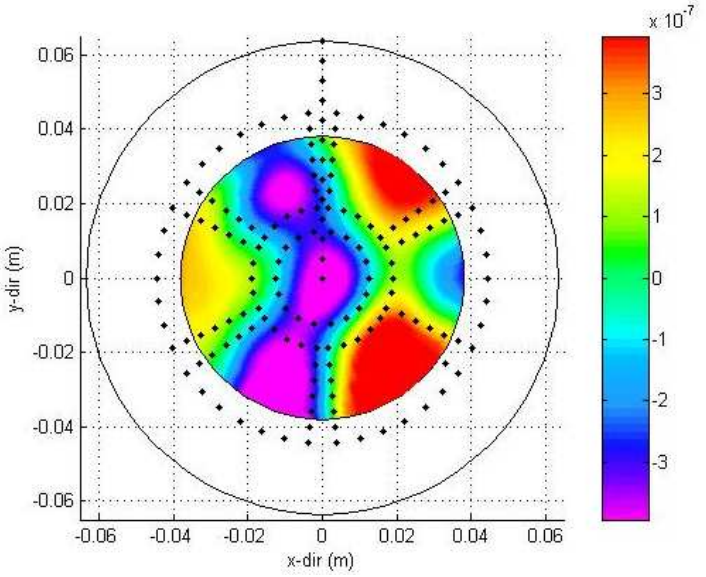

Figure 39: Obtained Surface in Closed-Loop Test One

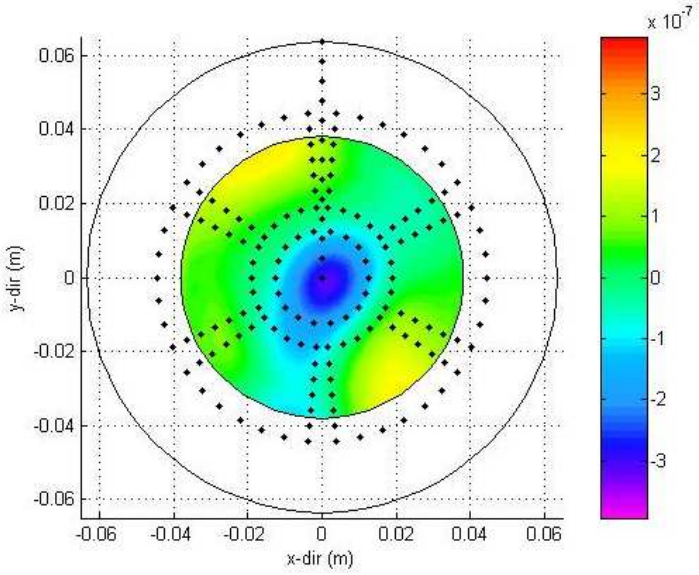

Figure 40: Average Residual of Closed-Loop Test One

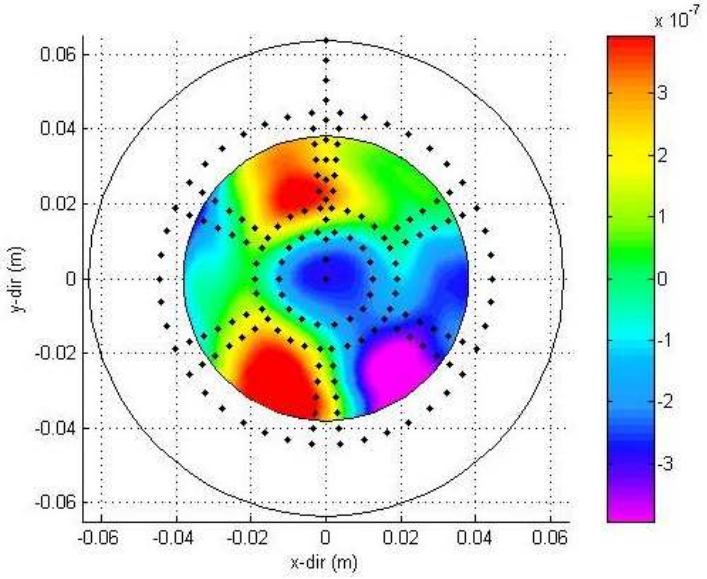

Figure 41: Obtained Surface in Closed-Loop Test Two

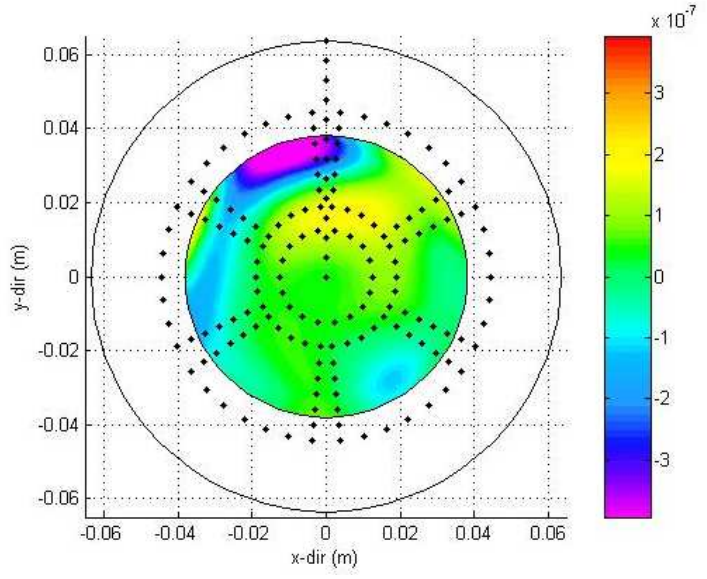

Figure 42: Average Residual of Closed-Loop Test Two

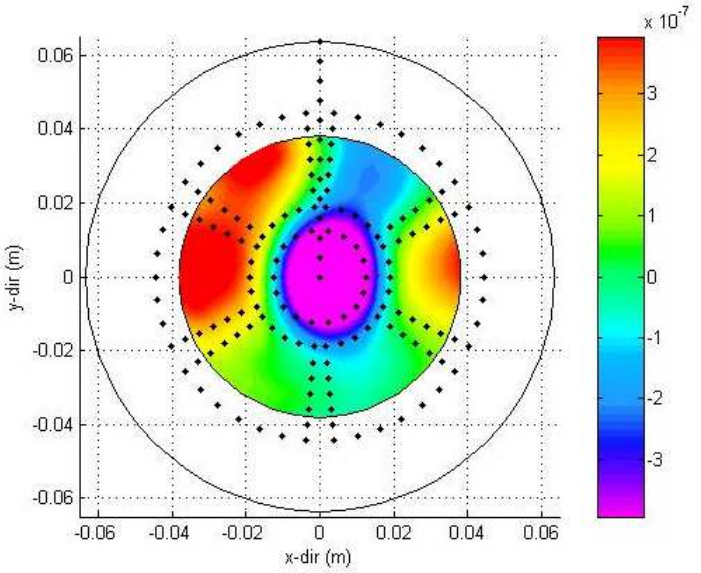

Figure 43: Obtained Surface in Closed-Loop Test Three

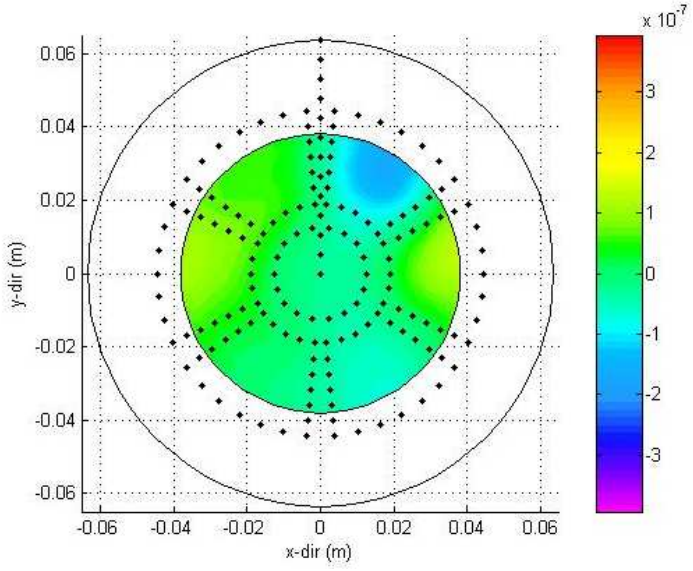

Figure 44: Average Residual of Closed-Loop Test Three

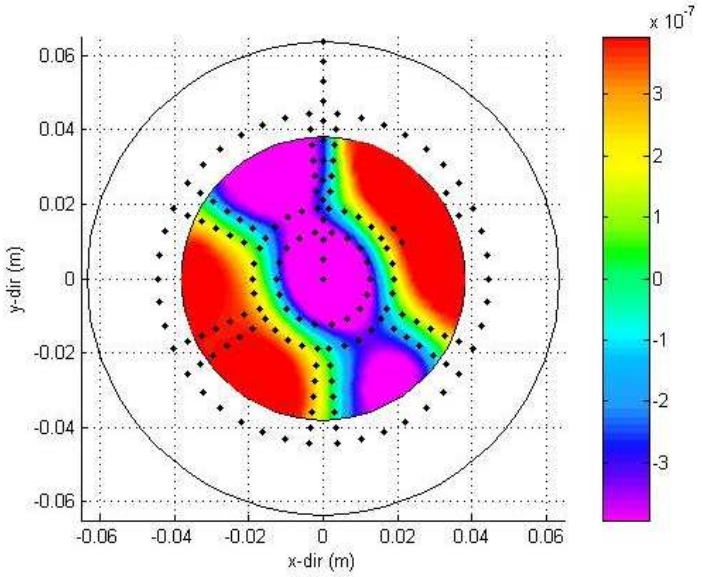

Figure 45: Obtained Surface in Closed-Loop Test Four

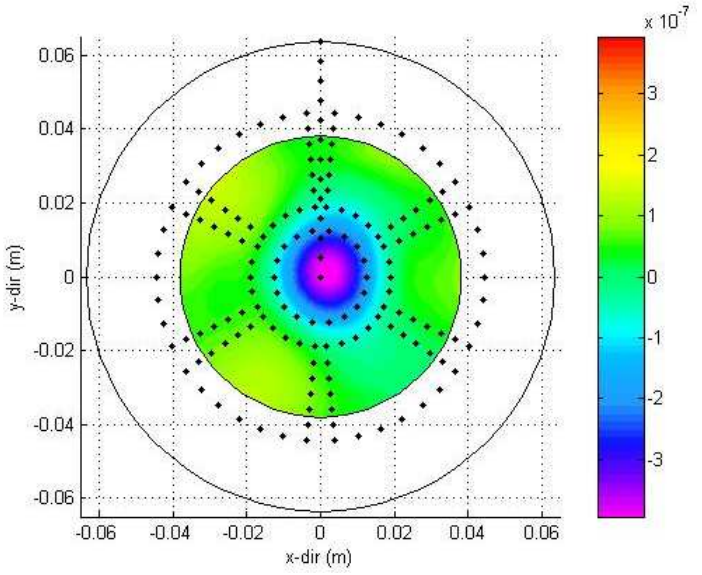

Figure 46: Average Residual of Closed-Loop Test Four

Z vectors recorded over a 20 second period from the time that the actuated surface reaches steady-state. The surface scale for each plot is given in meters. These plots show that the main features of each recognizable pattern in present. For example, in Control Test One, despite a significant residual error in the actuator one region, the ripple effect is evident around the mirror.

The residual error in the surface deflection of each test is captured in Figure 47. For each test, the negative magnitude of the residual error is plotted verses time. The residual error, R_e is calculated by Equation 13 where Z=[z_1, z_2, ...,z_{42}] is the obtained Zernike coefficients.

$$R_e = [\sum_{i=1}^{42}(Z_i - ZE_{expectedi})^2]^{\frac{1}{2}} \tag{13}$$

From the R_e plots, several characteristics can be measured and are summarized in Table 10. In the table, R_e is the average of the steady-state residual error calculated for each Z over a 20 second period, t_s is the time it takes to reach steady-state R_e for the first time, STD_{R_e} is the standard deviation of steady-state R_e, and STD_{Rest} is the standard deviation of the residual before feedback starts. Notice that for all four tests, the surface variation, captured in STD_{R_e} and STD_{Rest}, is on the same order of magnitude before and after feedback begins. However, the STD_{R_e} during Control Test Two is an order of magnitude larger than the other three tests and previously reported surface variation. It is likely the mirror was being excited by something else during the test or that something was causing additional noise in the wavefront measurement.

Table 10: Closed-Loop Test Result Summary

Test	t_s seconds	R_e μm	STD_{R_e} μm	STD_{Rest} μm	0.3 $D_{ZEexpected}$ μm
1	3	0.241	0.006	0.005	0.216
2	1.6	0.398	0.093	0.065	0.210
3	1.3	0.155	0.006	0.0034	0.165
4	0.7	0.245	0.010	0.018	0.330

Overall, the four closed-loop control tests demonstrate that mirror is controllable at a rate of 9.4 Hz in the feedback control system. However, Table 10 and Figure 47 show that expected results were not achieved. The steady-state error was expected to be less than a third of the surface deflection of $ZE_{expected}$. This value is included in Table 10 and shows that the steady-state error is near this maximum for all four tests. It was also expected that t_s would be less than one second. However, three of the four tests did not meet this. Finally, all of the tests exhibit significant overshoot.

There are several reasons for the difference in expected results. First, it is noted that there is a significant uncertainty in t_s caused by the method used to start the feedback loop. Feedback is started by sliding a digital slider in ControlDesktop to switch the gain from zero to one. Since this is manually done with a PC mouse, the time it takes to move your hand over will affect t_s. To correct

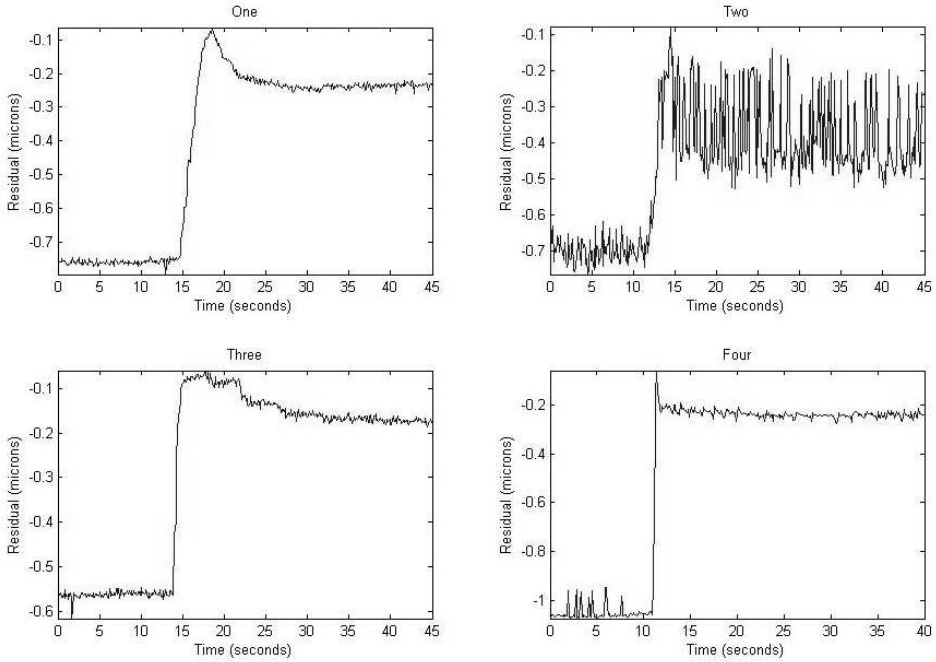

Figure 47: Residual Error in Four Closed-Loop Control Tests

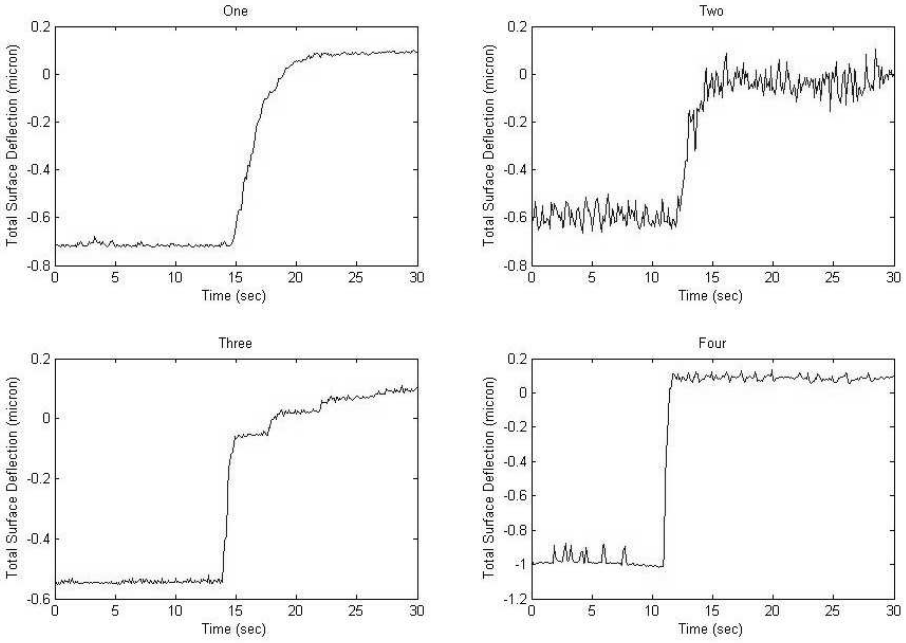

Figure 48: Total Surface Deflection Error in Four Closed-Loop Control Tests

this in future control tests, either an automatic switch should be found or the gain value should be recorded and accounted for.

Second, the steady-state error was significantly larger than expected because the the residual error reached a minimum peak and then increased to a reach steady-state. The reason for this is shown by Figure 48. This plots the total surface deflection during each test subtracted from the total surface deflection of $ZE_{control}$. Notice that the difference in the total surface deflection exceeds zero at the exact same time that the residual starts to increase in Figure 47. Prior to this point, the error was adding to the actuator command. As the error reverses direction, the actuator command approaches open-loop control.

Although this explains why the controller sent the actuator commands to open-loop control values, the control algorithm provided by Fernandez should have prevented the surface deflection from exceeding the control surface deflection. Upon further inspection of the control method, it was discovered that the reverse of his error equation was implemented. Therefore, the expected results were not based on correct assumptions. The error should be calculated as shown below.

$$e = ZE_{control} - Y_{(k-1)}$$

$$V_k = 0.3\ K\ e + K\ ZE_{control} = K(-0.3\ Y_{(k-1)} + 1.3\ ZE_{control}) \tag{14}$$

This misinterpretation could have been realized by looking at the latter part of Equation 11. Here a third of the measurement is adding to the actuator command. This assumes that the measurement will approach $ZE_{control}$. However, only 70 percent of $ZE_{control}$ is commanded so it is not feasible to assume the measurement will reach this with plant uncertainty and noise in the closed-loop system. By reversing the error as shown above, the measurement is subtracted to prevent the total surface deflection from exceeding the surface deflection of the expected surface.

To show that the expected results are possible by implementing the modified control equation (see Equation 14) with the reverse error, a final control test was executed. The results are shown in Figure 49. In this test R_e is 0.1385 μm, or 15 percent of the expected surface deflection, and t_s is 0.4 seconds. However, this controller caused an order of magnitude increase in the surface variation during feedback. Here STD_{R_e} was 0.034 μm but STD_{Rest} was only 0.004 μm. This variation during steady-state was not reported by Fernandez because the iterations were stopped once steady-state was achieved. Each sharp peak variation in the residual error plot does correlate to a point where the total surface deflection exceeds zero. Further characterization of this control method would likely find that it has an inherent problem dealing with the error in H and measurement noise and aliasing.

Implementing a more advanced control scheme would correct for this problem and is recommended for further research efforts.

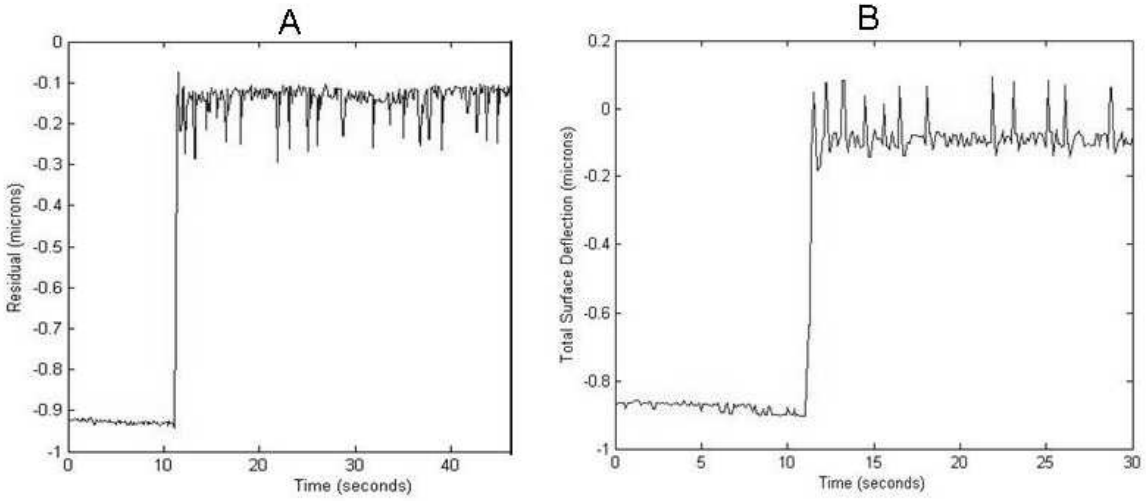

Figure 49: Reversed Error Control Test: A.Residual Error B.Total Surface Deflection

4.6 Low Frequency Disturbance Control Test

Now that a feedback control system has been demonstrated, a logical step is to examine its capability. This will help characterize the system and the test mirror for future control method designs. This section describes the results from introducing a low frequency disturbance into the closed-loop feedback system. The original feedback control method is not altered in any way for this test. Additionally, only two tests are run with the $ZE_{control}$ surfaces One and Two. Results are captured in time history plots of the residual error and surface deformation during the control tests.

4.6.1 Set-Up. The closed-loop feedback system created in this research has a control-update rate of 9.4 Hz. In order to observe the disturbance, the system must be at least twice as fast as it. In order to react to the disturbance, control updates are needed much faster than that. For this reason, a $\frac{1}{2}$ Hz disturbance was used to allow 18 control updates between each period. This is expected to allow the actuators enough time to dampen the disturbance.

The amplitude of a $\frac{1}{2}$ Hz disturbance in the mirror surface should be less then 0.3 μm in order not to exceed the actuator deformation capability and to minimize the affect on the wavefront measurement. To create the disturbance, a large piezo stack manufactured by Piezo Systems Inc. was used [28]. The piezo stack was set as one of the three blocks suspending the test mirror on

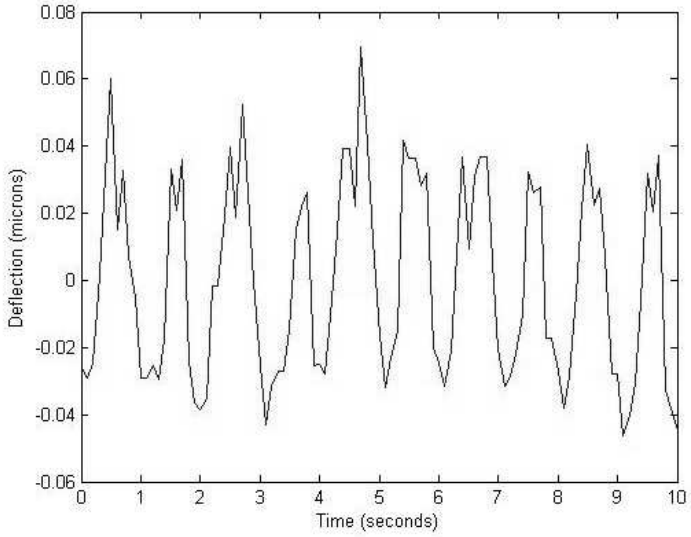

Figure 50: Total Mirror Surface Deformation While Piezo Stack is Actuated with Half Hz Sinusoid

the optical table. The stack is 18 mm high and 10 mm in width and length. The piezo stack is characterized to rise for a given voltage. By biasing the stack with an initial 40 volts, the stack could rise and fall with a sinusoid input.

A VH300 Laser Doppler Vibrometer was used to verify the piezo stack's movement when a $\frac{1}{2}$ Hz sinusoid biased by 40 volts was applied. The mirror was suspended on the stack in the same manner as for control testing and the displacement of the mirror frame above the piezo stack was measured. The amplitude of the sinusoid was incremented up to 20 volts to a capture how much amplitude was needed to move the mirror frame a certain amount. The test parameters are listed in Table 11. It was expected that displacing the mirror frame about 0.3 μm would cause a similar response in the mirror surface.

Table 11: Table of Test Parameters

Parameter	Setting
Switch	↑↑↑
Tspan	5 seconds
Fspan	80 Hz
Lines	400
dF	200.0 mHz
dT	4.883 ms
Overlap	None
Window	Flat
Block Size	1024

The results for a 10 volt amplitude is shown in Figure 51. Here, the $\frac{1}{2}$ Hz sinusoid is clearly transmitted to the mirror frame. The same transmission was evident for all amplitudes above 8 volts. Below 8 volts, the measure of displacement was not clear and showed significant variation between tests. It was determined that the amount of movement in the frame could not be measured accurately by this method because the resolution of the displacement measurements could not be enhanced. Additionally, Figure 51 shows that a 10 volt amplitude displaces the mirror frame 4.5 μm. It was expected that an amplitude less then this would create the desired 0.3 μm displacement in the mirror surface and would be determined with wavefront measurement.

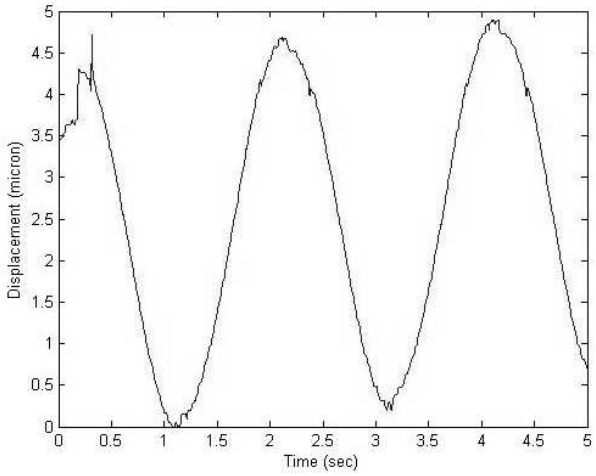

Figure 51: Mirror Frame Displacement While Piezo Stack is Actuated with Half Hz Sinusoid

Prior to the control tests, the reaction of the mirror surface to the piezo stack actuated with a $\frac{1}{2}$ Hz sinusoid biased at 40 volts was determined. In this case, the amplitude of the sinusoid was incremented up to 10 volts. The results were significantly different then expected. It took an amplitude of 10 volts to displace the mirror surface only 0.1 μm as shown in Figure 50. Below this amplitude, no significant disturbance is reflected in the wavefront measurement. Also notice that the disturbance looks like a One Hz period signal. This is a result of exciting other frequencies in the mirror and not a record of the piezo stack's movement. It was determined that a 10 volt amplitude would be used to move the piezo stack during low frequency control tests.

Each of the two low frequency control tests were run similar to the four closed-loop control tests with a few differences. Feedback was started within 10 seconds of the start of data recording. The closed-loop system was allowed at least 10 seconds to reach steady state. The $\frac{1}{2}$ Hz signal was turned on and sent to the piezo stack. After 20 seconds, the feedback was turned off to capture the disturbance without closed-loop control. Throughout each test, the 40 volt bias remained on the piezo stack and is included in the bias of the mirror and subtracted from all tests.

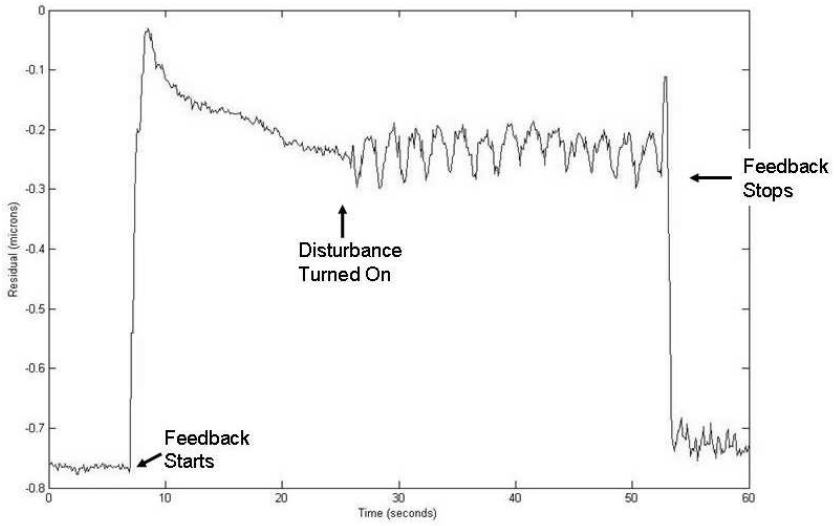

Figure 52: Residual Error of Low Frequency Disturbance Test 1

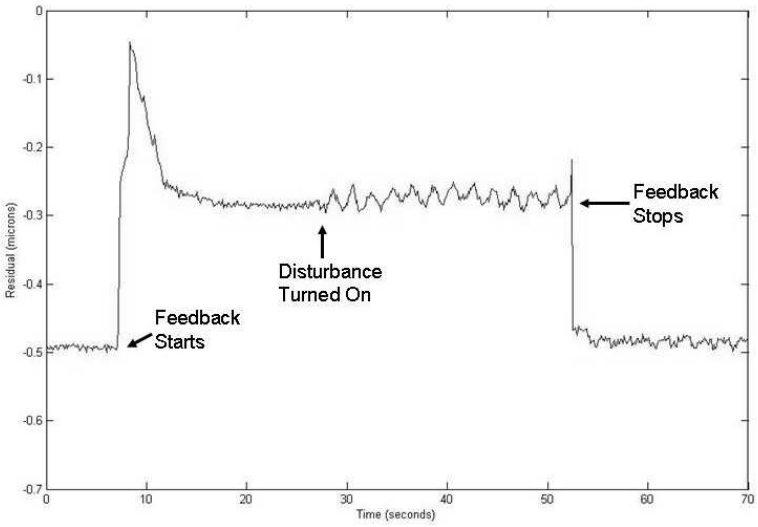

Figure 53: Residual Error of Low Frequency Disturbance Test 2

4.6.2 Results. The results are plotted in Figures 52 and 53. In Test One, the first 25 seconds are nearly identical to the closed-loop Test One. Here the t_s is only 1.1 seconds and R_e is 0.21 μm. Once the disturbance is started, it is easily tracked in the surface deformation. Once feedback stops, intermediate peaks show up and the magnitude of the disturbance is reduced. Additionally, the magnitude of the disturbance is reduced after feedback stops. In Test Two, the first 25 seconds are nearly identical to the closed-loop test Two. Here the t_s is 1.3 seconds but R_e is 0.286 μm. Additionally, the large surface variation observed in closed-loop Test Two is not present. Again, the disturbance significantly changes in magnitude and frequency once feedback stops.

The results show that the disturbance created was not controllable with the test set-up. This is a result of exceeding the capability of the feedback controller. As shown in the first 25 seconds of these two tests, it takes over a second for feedback control to reach the steady state surface. This means that this feedback control system can not update the control 18 times between each period with a $\frac{1}{2}$ Hz disturbance but only twice. This is not enough to expect to see any significant change in the disturbance.

Additionally, the capability of the actuators may have been exceeded also. The lag in the actuators found in the linearization calibration curves was large with respect to the 20 second hold in each step. Initial actuator commands in the closed-loop control tests change the actuator commands 100 times faster than this. This means that the lag in the actuators may prevent the mirror from reacting fast enough to a $\frac{1}{2}$ Hz disturbance. One way to determine if the actuators have the ability to control the $\frac{1}{2}$ Hz disturbance is to make the actuators recreate a 0.1 μm tilt with a period of 2 seconds. If the actuators can create the disturbance, they should be able to control it. Future disturbance testing should first create the disturbance by actuating the test mirror to ensure it is controllable. Additionally, further characterization of the relationship between the lag in the actuators and the amount of time between each step is needed.

Finally, the uncertainty in the piezo stack performance may have caused SHWS to exceed its calibration range. The ten volt amplitude causes the mirror frame to displace 4.5 μm as shown in the vibrometer test. This is a change of almost seven wavelengths which is greater than the SHWS can accurately measure. It was assumed that lower magnitude signals to the piezo stack would displace the mirror frame less. However, this could not be confirmed by the vibrometer or by wavefront measurement. One possibility is that the piezo stack does not displace the mirror at all when signal amplitudes are significantly less than the bias voltage. If this is the case, the disturbance used in this testing may also displace the mirror surface by 4.5 μm and the wavefront observed was not accurate. A way to characterize the piezo stack or a new way to create a disturbance is needed. This is an area that should be explored in further research.

4.7 Summary

In this chapter, the deflection characteristics of the test mirror is captured and closed-loop feedback is demonstrated with the data acquisition system. The measured IFS varied slightly between each actuator due to the material properties of the PVDF and large deflection variations that were possibly a result of manufacturing. The maximum IFS deflection was 0.5 μm. During measurement of the IFS, a 10 percent hysteresis was shown in the test mirror. Four surfaces were created with an open-loop method and evaluated. Their evaluation demonstrated the method used to capture the IFS needs to be modified to capture more of the mirror's nonlinear properties and get a better understanding of what is occurring. The four surfaces were then generated with the closed-loop control method to within 35 percent of their expected surface. Additionally, reversing the error in the control scheme achieved a surface within 15 percent of the expected surface. A low-frequency disturbance was added by moving one of the mirror's support legs. This disturbance caused excitation of other frequencies in the mirror's surface and could not be controlled because the controller implemented was too slow. Future efforts should be made to ensure the disturbance is controllable with the test mirror and either a faster controller or a slower disturbance is needed.

V. Conclusion

5.1 Overview

The objective of this research was to demonstrate closed-loop feedback control on an in-plane actuated deformable mirror to further asses its capability to maintain optical tolerances in a changing space environment. As a result, this project contained two distinct steps that were needed to further characterize the mirror. First, a new data acquisition system was developed to increase the system rate and provide a method of feedback from the mirror surface. Second, open-loop and closed-loop tests were set-up to utilize the data acquisition system and characterize the mirror.

The data acquisition system incorporates a data conversion and control model, a wavefront measurement sensor, and the in-plane actuated deformable test mirror. The models are critical to receiving feedback from the mirror surface and implementing a control method in a closed-loop system. The data acquisition model sets the overall system rate. The wavefront sensor can measure micron level distortions in the mirror's surface and export that information in 42 Zernike coefficients. Finally, the test mirror has seven actuators, each of which can be actuated by the analogue output from the control model.

Control tests were then developed utilizing the data acquisition system. First the response characteristics of the test mirror were captured by stepping the actuator commands through their operational range. Then open-loop control tests demonstrated how well the linear approximation for the response characteristics represented the mirror and characterized the static deformation of the mirror. Closed-loop control tests presented a simple control scheme using feedback control with the data acquisition system. Test set-up and results are provided.

5.2 Conclusions Drawn

A new data acquisition and control system was developed for use with AFIT's new wavefront imaging hardware and optical table set-up. The system achieved an operating rate of 9.4 Hz that could run for 90 seconds or a rate of 9.1 Hz that could run indefinitely. This is three times faster than a previous system used at AFIT but still not sufficient to capture or control the dynamics of the test mirror. Limiting factors were identified as the data transfer and conversion between SHWS and ControlDesktop. The biggest advantage that the data acquisition system provided is the ability to create actuator commands via a signal block or with a control model in Simulink®. Simulink® is a common commercial controls software that can support many control methods.

The relationship between the actuator commands and their effect on the surface deformation were characterized in a linearization test. Unexpected deflection jumps of 0.1 μm and a hysteresis of 10 percent were captured in the calibration curves. Calibration curves also showed that each actuator

could positively deform the mirror between 0.28 μm and 0.51 μm. This was up to 100 percent more than predicted in an FEM. Differences are noted as un-modelled manufacturing and nonlinear properties. Using a single linearization test is limited because additional response characteristics with respect to actuation speed and magnitude could not be captured. Capturing all the the nonlinear properties in the mirror should be done with multiple tests.

Open-loop control tests were performed utilizing the response characteristics captured in the IFS. The open-loop actuator commands were an average of 225 volts different than the desired actuator commands and the average residual error of the open-loop surfaces was 0.36 μm. This suggested that the nonlinearity of the mirror significantly affects the IFS. Incorporating a better representation of the nonlinearities may improve them. Open-loop tests were also used to calculate a 6 percent surface measurement variation. This is the expected error during all tests. Finally, open-loop control tests demonstrated that recognizable surface shapes were easily repeatable with the test mirror.

Closed-loop control tests implemented a simple iterative controller around the IFS where a third of the error is added at each iteration. This demonstrated that the expected surface could only be corrected to within 30 percent in three seconds. It was determined that the controller was implemented in a reverse scheme than intended. Implementing it as intended demonstrated that the expected surface could be corrected to within 15 percent in 0.4 seconds. Either control scheme is not ideal for real-world application but showed closed-loop control is feasible with the data acquisition system. Finally, the control method implemented was not fast enough to provide any control to a $\frac{1}{2}$ Hz disturbance. It is also not certain that the test mirror can minimize this type of disturbance for any control implementation using the current measurement set-up.

5.3 Areas for Further Development

Many areas of further study were noted during this research. Four specific areas would significantly enhance efforts to characterize the mirror. These are described below.

First, to further explore the mirror's dynamic characteristics, the system measurement rate must be increased. This would involve either improving a combination of parts with data rate limitations in the current feedback system or obtaining a new system. The first solution could include updating the RS232 dSpace® serial port to RS422 and changing the Nport® converter box to one that has a faster baud rate. Just changing to a RS422 port will only increase the system rate by a factor of 2 because the Nport® device has a maximum baud rate of 230400 $\frac{bits}{second}$. The maximum baud rate of the RS422 port is one million $\frac{bits}{second}$. The third part that would need to be changed is the data conversion model. The binary format carries the data most efficiently, but

an efficient model was not created for it. This would include converting the binary data in c-coded S-Function blocks and receiving the data as a byte-packet. These three changes would increase the baud rate up to one million $\frac{bits}{s}$ and equate a system rate of 370 Hz. To reach data rates higher than this, obtaining a new system for feedback control should be explored.

Before upgrading the current system would be a viable solution, the data rate limitation at 34.5 Hz needs to be determined. The limitation is either in the SHWS script exporting the data or in the serial receive hardware. This turn around time for the serial receive hardware can be monitored once a serial interrupt block is incorporated into the data conversion model.

Second, this research laid the groundwork to implement a control algorithm in a Simulink® model. The control equation presented used a 0.3 factor that provided a stable closed-loop response in the test mirror. The reason other values adversely affected the closed-loop response needs to be further explored because the mirror can only perform as well as the control scheme allows. This also suggests that implementing advanced control methods will demonstrate more potential in the capability of the mirror. This would involve incorporating a nonlinear block to remove the effects of hysteresis and implementing a control scheme that does not ensure steady-state error or a significant rise time. The method to start feedback should be modified to reduce any affect on t_s. Additionally, utilizing a disturbance that is known to be controllable is needed. This involves using one that is slow enough for the control update rate and one that can be created with the actuators. These two steps are vital to demonstrating suitability for a space application.

Third, in order to implement a nonlinear controller, the nonlinear response in the mirror needs to be further characterized. This may be done with multiple linearization tests. Varying the volt-step time will capture how time-dependant the hysteresis is. To implement faster volt-steps, the low-pass filter on the step signal should be adjusted to be consistent with the sampling frequency of the system. Also, the effect of varying the volt-step amplitude will characterize what happens when a large actuator command is suddenly sent to an actuator.

An effort to reduce the nonlinearities in the mirror which may be introduced during manufacturing should also be explored. This will improve the linear approximation. Possible ways to reduce nonlinear effects involve creating an even amount of silver paint over each lead to reduce capacitance differences. This could be done by evaporating a thin silver layer on the mirror in the vacuum chamber before the actuator pattern is etched. This would remove the need to fix the leads. Additionally, the Piezo Sensors company which manufactures the PVDF material, notes that the adhesive in copper tape, which is used to attach wires to the leads, acts as a dielectric and may cause additional capacitance to the actuator. Therefore, all lead attachments should be identical to each

other. Finally, new types of PVDF material that are not susceptible to breaks are being developed and could be used in future mirrors.

Finally, the characterization and control of the test mirror was limited in this projected by the fidelity of the actuators. Manufacturing methods to create a detailed actuator pattern on the PVDF material have been explored. Increasing the number of actuators and updating the control system to incorporate all of them would greatly enhance control tests and comparisons to other deformable mirrors.

Appendix A. Laboratory Notes

A.1 Directions to Use 1225-Byte Model

A.1.1 Set TCL Scripts on WaveScope PC.

```
In the C drive, under the path \usr\aos\wavescope\src\scripts\TestEx.tcl find the following lines:

If {$remote_sock != } {
     set dmpstr [a.dump ws_results(Zernike)]
     set ascstr [resub all {[(<>\n)]} $dmpstr ]
     set binstr [binary format f84 $ascstr]
           puts $remote_sock $binstr

and if necessary replace with:

If {$remote_sock != } {
     puts $remote_sock [a.dump ws_results(Zernike)]

In In the C drive, under the path \usr\aos\wavescope\src\scripts\Socket.tcl find the following lines:

fconfigure $newSock blocking 0
fconfigure $newSock encoding binary

and if necessary replace with:

fconfigure $newSock translation {auto crlf}
```

This will configure the TCL scripts to export the data in ASCII characters.

A.1.2 How to Collect Data.

1. Set mirror in optical set-up with reference and test reflections centered.
2. Turn on WaveScope® Ebox and start software. Choose MLM, align, and calibrate.
3. Start Matlab® on the Dell® PC connected to dSpace®. Set directory under the C drive to /Documents and Setting/Administrator/Desktop/Matlab/Marcum-1225 byte model.
4. Turn on dSpace®.
5. Open ControlDesktop on the Dell® PC connected to dSpace®. Click on platform tab in navigator control bar. In file sector tab at the bottom of the screen, find $Serial\_1225.sdf$ by path under the C drive Documents and Settings/Administrator/Desktop/Matlab/Marcum-1225 byte model. Drag and drop file in ds1103 in platform tab.
6. Open layout $layoutzern$ under same directory and click yes to load data connections.
7. Open $Serial\_1225.mdl$, set-up and build the model. Make sure to run AlwaysRun.m to set matrix A and arbitrary numbers for parameters K, Ze0, and Zecontrol in the model. Note that Ze0 will be set by Ze0.m after a data set is captured, K will be set by running a linearization test and DataAve4.m, and Zecontrol is set for a desired test.
8. To run a control test, a slider gain may be desired to turn on and off the actuator control signals. To do this, choose a slider gain gauge from the instrument selector controlbar window and set it in the current layout. Under the tab next to the file selector, in the model root directory, find the Gain and click it. Drag and drop the boxed p in to the gauge. This allows you to change the value during simulation.
9. Under the capture settings controlbar, set run time, down-sampling, and uncheck auto-repeat.
10. Return to Matlab®. Build the model again. (Make sure WaveScope® is paused). As soon as the model starts compiling, start WaveScope® live display and then Control Desktop Animation mode immediately. This model is limited to about 90 seconds.

A.1.3 How to Change the Number of Zernike Coefficients.

1. Change number of Zernike coefficients being sent by SHWS. Under the execute menu, choose change parameters and change the number of Zernike coefficients.
2. In the 1225-byte model change several of the blocks to reflect new number of bytes being received. Total bytes is equal to number of Zernikes multiplied by 29 bytes + 7 bytes. Here is an example of how to change the model to receive 10 coefficients, or 297 bytes:

- In the $DS1103SER\_RXBYTES$ block: Under the rx parameters tab, set the number of bytes to 297.
- In the $S\text{-}function\_Builder2$ block: Under the data properties tab, change the number of columns for the input and output ports to 297. Under the output tab, change the variable bytes to 297.
- In the $S\text{-}Function\_Builder3$ block: Under the data properties tab, change the number of columns for the input and output ports to 297 and 10. Under the output tab, change the third line of code to have a 10.
- In the $Zernike\_Access$ block: Change the mux to only have 10 outputs.
- Change the model variables: A and K should be 10 by 10. Ze 0 and $Ze\ Control$ should be 10 by 1.

A.1.4 When to Use 1225-Byte Model.

- Acquisiton speed determination tests
- Surface control tests, open and closed-loop.

A.2 Directions to Use 1-Byte Model

A.2.1 Set TCL Scripts on WaveScope PC.

```
In  C:\usr\aos\wavescope\src\scripts\TestEx.tcl  find the following lines:

 If {$remote_sock != } {
     set dmpstr [a.dump ws_results(Zernike)]
     set ascstr [resub all {[(<>\n)]} $dmpstr ]
     set binstr [binary format f84 $ascstr]
           puts $remote_sock $binstr
}
 and if necessary replace with:

If {$remote_sock != } {
     puts $remote_sock [a.dump ws_results(Zernike)]
}
 In C:\usr\aos\wavescope\src\scripts\Socket.tcl find the
following lines:

fconfigure $newSock blocking 0
fconfigure $newSock encoding binary

and if necessary replace with:

 fconfigure $newSock translation
{auto crlf}
```

This will configure the TCL scripts to export the data in ASCII characters.

A.2.2 How to Collect Data.

1. Set mirror in optical set-up with reference and test reflections centered.
2. Turn on WaveScope® Ebox and start software. Choose MLM, align, and calibrate.
3. Start Matlab® on the Dell® PC connected to dSpace®. Set directory under the C drive to /Documents and Setting/Administrator/Desktop/Matlab/Marcum-1225 byte model.
4. Turn on dSpace®.
5. Open Serial2.mdl, set-up and build the model. Make sure to run AlwaysRun.m to set matrix A and arbitrary numbers for parameters K, Ze0, and Zecontrol in the model. Note that Ze0 will be set by Ze0.m after a data set is captured, K will be set by running a linearization test and DataAve4.m, and Zecontrol is set for a desired test.
6. Open ControlDesktop on the Dell® PC connected to dSpace®. Click on platform tab in navigator control bar. In file sector tab at the bottom of the screen, find Serial2.sdf by path: /Documents and Settings/Administrator/Desktop/Matlab/Marcum-1 byte model/. Drag and drop file in ds1103 in platform tab.
7. Open layout *layout* plotter under same directory and click yes to load data connections.
8. To run a control test, a slider gain may be desired to turn on and off the actuator control signals. To do this, choose a slider gain gauge from the instrument selector controlbar window and set it in the current layout. Under the tab next to the file selector, in the model root directory, find the Gain and click it. Drag and drop the boxed p in to the gauge. This allows you to change the value during simulation.
9. Under the capture settings controlbar, set run time to 1100 seconds, down-sampling to 1000, and uncheck auto-repeat for linearization test.

10. Return to Matlab®. Build the model again. (Make sure WaveScope® is paused). As soon as the model stops compiling, start WaveScope® live display and then Control Desktop Animation mode immediately.

A.2.3 When to Use 1-Byte Model.

- Linearization tests
- General shape and Zernike limitation tests

A.3 Directions to Use 8-Byte Model

A.3.1 Set TCL Scripts on WaveScope PC.

```
 In  C:\usr\aos\wavescope\src\scripts\TestEx.tcl  find the following lines:

If {$remote_sock != } {
     puts $remote_sock [a.dump ws_results(Zernike)]

} and if necessary replace with:

 If {$remote_sock != } {
     set dmpstr [a.dump ws_results(Zernike)]
     set ascstr [resub all {[(<>\n)]} $dmpstr ]
     set binstr [binary format f84 $ascstr]
           puts $remote_sock $binstr
}
 In C:\usr\aos\wavescope\src\scripts\Socket.tcl find the following line:

 fconfigure $newSock translation {auto crlf}

and if necessary replace with:

 fconfigure $newSock blocking 0
fconfigure $newSock encoding binary
```

This will configure the TCL scripts to export the data in binary format.

A.3.2 How to Collect Data.

1. Set mirror in optical set-up with reference and test reflections centered.
2. Turn on WaveScope® Ebox and start software. Choose MLM, align, and calibrate.
3. Start Matlab® on the Dell® PC connected to dSpace®. Set directory under C drive as /Documents and Setting /Administrator/Desktop/Matlab/Peterson-Binary model.
4. Turn on dSpace®.
5. Open $Serial\_Binary.mdl$, set-up and build the model. Make sure to run AlwaysRun.m to set matrix A and arbitrary numbers for parameters K, Ze0, and Zecontrol in the model. Note that Ze0 will be set by Ze0.m after a data set is captured, K will be set by running a linearization test and DataAve4.m, and Zecontrol is set for a desired test.
6. Open ControlDesktop on the Dell® PC connected to dSpace®. Click on platform tab in navigator control bar. In file sector tab at the bottom of the screen, find $Serial\_Binary.sdf$ by path under C drive: /Documents and Settings/Administrator/Desktop/Matlab/Peterson-Binary model/. Drag and drop file in ds1103 in platform tab.
7. Open layout $layout42zerns$ under same directory and click yes to load data connections.
8. To run a control test, a slider gain may be desired to turn on and off the actuator control signals. To do this, choose a slider gain gauge from the instrument selector controlbar window and set it in the current layout. Under the tab next to the file selector, in the model root directory, find the Gain and click it. Drag and drop the boxed p in to the gauge. This allows you to change the value during simulation.
9. Under the capture settings controlbar, set run time.
10. Return to Matlab®. Build the model again. Start WaveScope® live display and then Control Desktop Animation mode.

A.3.3 When to Use 8-Byte Model.

- Acquisition speed tests
- Surface control tests, open and closed-loop

A.4 Testing and Repairing Leads

A.4.1 Testing Leads.

Don Halvorsen a director at MSI Sensors suggested the best way to test for good leads is to check the capacitance. The capacitance of 52 micron PVDF is about 1.31 $\frac{nf}{in^2}$. Use the Dynascan Corporation 820 capacitance meter and connect between ground and the end of a lead. A good patch has a capacitance between 1.3 and 1.9 nf (see Reference [8]).

A.4.2 Fixing Leads.

1. Determine where the etched electrode lead is bad by moving the positive end of the capatance meter along the lead until you get a good reading.
2. Paint a very thin layer of silver paint from the edge of the lead to just past the spot where the lead is broken.
3. Allow paint to dry and measure capacitance again.
4. If necessary, silver paint thinner can be used to remove the silver paint.

A.5 Data Acquisiton Set-Up by Andrew Marcum

This is a paper on the description of data acquisition set-up by intern Andew Marcum, summer 05. Figures have been eliminated but are not needed for comprehension.

Title: Wavescope Data Acquisition with dSPACE

Andrew Marcum

Rose-Hulman Institute of Technology

Electrical Engineer 06

Air Force Institute of Technology

Summer Intern 05

22 August 2005

Introduction

For successful determinat on of deformable mirror surface flatness, Wavescope must provide a set of Zernike coefficients calculated by analyzing laser light reflected from a test mirror. Zernike coefficients provide important information concerning mirror surface flatness; values for creating computer generated 3-dimensional plots of the mirror surface, and provide a means for closed loop control. The Wavescope is setup in such a way that during a test in which real-time data (Zernike coefficients) is measured from the mirror, the data is routed digitally to an Ethernet port and output from the Wavescope computer via Ethernet cable.

dSPACE Hardware Setup

The dSPACE PPC1103 sy tem was selected to receive the Wavescope data due to its MATLAB and Simulink capabilities. The dSPACE computer system is stocked with a UART (Universal Asynchronous Receiver/Transmitter) RS232 port which can be used to send or receive digital data as well as several analog to digital (A/D) and digital to analog (D/A) ports. Since the data is already in the digital form, A/D capabilities are not needed for this experiment. However, in order to control the mirror, analog signals must be looped back to mirror to create a desirable surface. This can be done using the D/A capabilities.

Because the raw data stream sent from Wavescope was done through Ethernet, the signal must be converted to RS232 or serial in order for the dSPACE system to read in the data. This was achieved by routing the Ethernet signal to a MOXA NPort Express DE-211. The NPort device simply converts the Ethernet signal to serial. The resultant serial signal can then be wired to dSPACE. In order for the NPort device to work properly, it must be setup in such a way so that the serial stream leaving the device can be read by dSPACE. Using MOXA software, the NPort device was initialized from the Wavescope computer. Figure 1 displays the serial properties set by the NPort device.

dSPACE Serial Setup

In order to read/write dSPACE I/O devices, a Simulink model can be created using a set of dSPACE coded s-function blocks. The Simulink Library Browser can be opened in MATLAB to provide access to all available Simulink blocks. Assuming the dSPACE software is correctly installed on the computer, the library browser will provide two tabs titled dSPACE MotionDesk Blockset and dSPACE RTI1103. The blocks needed to create the serial port connection can be found under the RTI1103 tab. The blocks of interest include the DS1103SER Setup (serial setup) and the DS1103SER RX (serial receive). Pictures of the blocks can be seen in Figure 2. The serial setup block provides a way to initialize the digital signal properties of the dSPACE serial port. In this case, the block settings were set to match exactly with that of the NPort device output signal. The appropriate properties can be found in Figure 1.

One additional parameter needed to complete the serial setup is the serial port buffer size. The buffer size must be a power of two as well as any value less than the max size of 524,288 bytes. For this experiment, 524,288 was selected in attempt to minimize any potential buffer memory error. The serial receive block offers many items useful for debugging such as access to output values, number of bytes received, or port reception status. The most important setting found in the serial receive block is the location where the user must set the number of bytes to receive.

Data Stream

The raw data stream trans itted from the Wavescope is in the form of ASCII characters. Therefore, in order to accurately read the Zernike coefficients, the ASCII characters must be decoded into a meaningful result. Each data byte or ASCII character sent from Wavescope is part of a 1,225 byte packet. Out of each packet of 1,225 bytes, each group of 29 bytes makes up one of the 42 coefficients. Extra data can be found in the form of 4 garbage bytes received at the beginning of the message and 3 garbage bytes received at the end of the 1,225 byte message. The end of each 29 byte Zernike coefficient contains the ASCII characters CR and a LF otherwise known as carriage return and linefeed. The purpose of the CRLF is to separate the ASCII characters into chunks which make up the Zernike coefficients. The rest of the ASCII characters used to make up the message are sent in the form of numbers, negative signs, brackets, and decimal points. Figure 3 displays a table containing the ASCII characters and decimal values used by Wavescope.

Simulink Model 1225 Byte Method

The first of two succ ssful attempts in receiving the Zernike coefficients was designed to read in the entire 1225 byte message in a single time step. This value was set in the serial receive block seen in Figure 2. Two c-coded s-functions were generated using the S-Function Builder block found in Simulink. The functions were created to do all the conversions from ASCII to decimal values. An image of the S-Function Builder block can be seen in Figure 2. The benefit of using the builder block rather than creating s-functions from scratch was due to the fact that the builder block generates the c-code layout for the s-function. With the layout, it is relatively easy to add the necessary code to implement a users desired task. Without the layout, the user must type the entire code from scratch. C-code can be added either through graphical user interface (GUI) under the outputs tab or by modifying the c-wrapper files generated by the builder block.

The first function generated for the 1,225 byte model contains a set of case statements which convert the ASCII characters to the numbers and symbols in which the character represents. For example, if the ASCII character 1 is read from the serial port, the user would actually see the decimal number 49 since 1 in ASCII is 49 in decimal. The case statements provide instructions which say that if the input is decimal 49, output a decimal number 1 in its place. This allows the ASCII character 1 to be converted to the number 1 in decimal. Case statements are provided for all possible ASCII characters sent from Wavescope. The values can be seen in Figure 3. The second function picks out the necessary numbers and symbols and organizes them into useful information. For example, the code utilizes counters and for-loops to search through the 42 chunks of 29 bytes to find the decimal point and surrounding numbers which create a decimal form of the Zernike coefficient.

The result of combining the dSPACE functions with the user created s-functions into a Simulink model can be found in Figure 4. The dSPACE system does not run entirely through Simulink. Basically, dSPACE uses Simulink with Real-Time Workshop to generate code which in turn operates on the dSPACE processor. In order to create the code from the Simulink model, the user must build the model. This can be done in Simulink or in dSPACE software called Control Desk. Upon completion of the build, the code is automatically linked to the dSPACE processor and the program begins to run.

An important setting found in Simulink is the step size. The step size is in units of seconds and is the time between dSPACE processor samples. For this experiment, the step size is set at .01 s or 100 Hz. Some important things to note from looking at Figure 4 are the D/A (DAC) blocks located at the output of the model. The DAC blocks are dSPACE coded blocks which provide the analog signal needed for closed-loop control of the mirror. The reason for the Cancel Dac Gain blocks is due to the fact that each DAC block is hard-coded to amplify the input signal by a magnitude of 10. Therefore, by multiplying the signal by 0.1 before entering the DAC, the gain associated with the DAC block is essentially canceled. The uint8 to double block converts the raw data stream from unsigned integer 8 to type double. This is necessary because the D/A blocks require the input to be a number of type double for the digital to analog conversion. The ASCII Converter block contains both the c-coded s-functions. The Zernike Access block provides access to the Zernike coefficients post conversion. To access this data, the dSPACE Control Desk software must be utilized. The Reference Zernikes and the DAC Selector blocks are used for purpose of real-time control of the mirror.

Problems with 1225 Byte Model

After running many tests with the Simulink generated 1,225 byte model, it was determined that the dSPACE system could not successfully handle the amount of data in which it was asked to receive. dSPACE was setup to read in 1,225 bytes at a rate of about 3.5 Hz. Over 4,200 bytes per second were received with this design. After approximately 20 30s of real-time data transfer, a buffer memory error would occur corrupting the data received from that point forward. This turned out to be extremely inconvenient for two reasons. The first reason is that we could safely only take 20 seconds of data which is not nearly enough to perform mirror control. The second reason is it was very time consuming and inefficient to stop the model,

save 20 seconds worth of data and rebuild. Because of these problems, a new solution was needed in order to successfully use the Wavescope data for mirror control.

Simulink Model 1 Byte Method

The second model created to read in the Zernike coefficients into dSPACE was designed to do so by reading one byte at a time rather than 1,225. This model can be seen in Figure 5. The advantage of reading only one byte from the buffer at a time is that this would prevent the buffer memory error since the buffer could easily handle reading a single byte rather than 1,225 bytes. However, because the user could only have access to one byte at a time, the overall complexity of the program increased. In order to correctly convert the 29 byte ASCII message, the user must have access to all 29 bytes. The reason for this is that the data signal is asynchronous and each packet of data is completely autonomous from one another.

For each packet of information sent, the code starts from beginning to end during data transfer and the operation is independent from all other operations. There is no way to keep track information from run to run without delaying bytes and storing them into memory. One way to store the bytes is to use the Tapped Delay block found in the Simulink Library Browser. The Tapped Delay block delays a signal for the duration of the model step size. In this case, the signal must be delayed 29 times in order to retain the necessary information of each coefficient. The output of Tapped Delay is a 1x29 vector with the eldest byte in the first place and the youngest in the vectors last place.

As bytes are cycled through the delay block, the vector must be ordered in a certain way so that the data can be converted into useful information. The ASCII Conversion block provides an s-function containing an if-else statement which allows the data to pass through the block only if the 1st byte in the vector matches with the 1st byte of the 29 byte message. The block also checks for the 3rd to last byte and confirms that it also matches 3rd to last byte in the 29 byte message. This method works for all cases since it has been determined from raw data stream testing that the 1st byte sent from Wavescope must be the character ¡ and the 3rd to last byte must be the ASCII character ¿ for cases in which useful information is received. Once the correct ordering of 29 bytes is passed through the filter, the bytes are converted to decimal values using the same set of case statements as in the 1,225 byte model.

At this stage, the signal of vector 29 must be aligned into the coefficient value. In the 1,225 byte model, the 1,225 byte signal is split into 42 chunks and converted to each of the 42 Zernikes. In this case, we have access to only 29 bytes at a time. Therefore, the output signal will start at the first Zernike and continuously cycle through all 42. Without some indication of which number Zernike is being output at a certain time, it is impossible to split up the data into sections of 42. Fortunately, the first half of the 29 byte signal contains the coefficient number while the second half contains actual coefficient. The coefficient number can be output alongside the coefficient value so that correct groupings of 42 coefficients can be created.

The Coefficient Number Organizer block receives the 29 byte output from the ASCII conversion block. This s-function block checks the decimal values of the first eight bytes and finds the value representing a decimal point. If the decimal point is found in the 7th place, the decimal value in the 6th place is the number and is output. If the decimal point is found in the 8th place, the 6th place multiplied by 10 and added to the number in the 7th place to create the value, which is output. The Zernike Organizer block contains an s-function which checks the last 16 bytes of the length 29 signal. The conversion to the Zernike coefficient is done similarly to the conversion of the Coefficient Number Organizer block. The s-function searches through the 16 bytes and finds the corresponding decimal values for minus signs and decimal points as reference and uses the surrounding values to build the coefficient number.

The last block needed to complete the data processing section of the 1 byte model is the Zernike Matrix Builder block. The Zernike Matrix Builder block contains 2 inputs and 42 outputs. The block basically checks the coefficient number and the corresponding Zernike value and routes that value to the block output number matching the coefficient number. For example, if the block receives the signal with coefficient number of 1 and Zernike value of .001, the code routes the .001 signal to output number 1 of the block. The code achieves this through a series of if statements and does this for all 42 coefficients. The code is necessary because the resultant output signal is a vector of all 42 Zernike coefficients, which is identical to that of the output of ASCII Converter block found in the 1,225 byte model. This means the mirror control section of the 1,225 byte model compared to that of the 1 byte model is exactly the same.

Problems with 1 Byte Model

Because of the 29 time delays used in the model, the processor speed of the dSPACE hardware must be increased. The step size must be small enough that it can do all the necessary delays and cycle through all the code in a rate faster than

the 3+ Hz raw data signal. If the step size is not small enough, the data will output slower than the data sent, thus loosing the real-time advantages of dSPACE. Therefore, the step size was modified several times until a value of 0.0001 seconds was chosen, which eliminated the delay problem. Another issue found with the 1 byte model was in the operation of dSPACE Control Desk. Due to the increased processor sampling speed, the data signals would be sampled upwards of 100+ times per coefficient when only a single sample is necessary. This generated giant data files which could not be easily processed with MATLAB or easily saved with Control Desk. Also, the Control Desk display could not update fast enough to see the data in real-time due to the heavy memory use of the sampled data. A solution was found in a setting named Downsampling found within Control Desk. The Downsampling feature allows the user to modify how many of the data samples to save in memory. For the 1 byte model, Downsampling was set to 1000. This means that the computer would save data for every 1000 samples. An easier way to think about this is that with Downsampling equal to 1000, the rate at which data is stored in memory changes from saving values every 0.0001 seconds to 1000 * (0.0001) seconds or 0.1 seconds. With the new rate, data is saved at 10 Hz, which is fast enough to not miss any coefficients or for aliasing to occur since the data stream is sent at approximately 3 4 Hz. This change drastically decreased the amount of data saved in memory and allowed the dSPACE Control Desk displays to successfully update with each set of 42 Zernikes.

Control Desk

The dSPACE Control Desk software must be implemented in order to save the real-time data. The Control Desk software allows the user to open a Simulink model and generate plots and figures associated with the variable states of the Simulink model. A variable description file (.sdf) is needed to load the Simulink variables. The .sdf is generated whenever the Simulink model is built or rebuilt. Another useful reason for using Control Desk is that all data designed to display in figures and plots can be saved to a .mat file and post processed with MATLAB. Also, Control Desk offers several digital displays, which can update in real-time, as well as sliders and buttons which allow for real-time changes to the Simulink model.

Appendix B. MATLAB Code

B.1 DataAve4

This code is used to process the data from the linearization test. It reads in each actuator test and was used to create hysteresis plots and calculate the control matrix.

```
%%%%%%%%
% Name: DataAve4.m
%%%
% This function reads in linearization test for each patch.  Each test is
% ran for 1000 seconds with 50 volt increases every 20 seconds.  The code
% then assumes the test is started as the model builds and takes 10 seconds
% of data between the steps.  The static reference is averaged between 5
% and 15 seconds and between 965 and 975 seconds at the end.

% Since the 1-byte model has a time lag, the time step is actually scaled
% by 1.1218.  For example, at 20 s the voltage is stepped to 0.25V but that
% data does not start to be recorded until 20*1.1218 = 22.4 seconds.
% Therefore, 1094 seconds of data is needed to capture every step.
% Do to uncertainties in the system, this is verified for each test
%
% patch 1 contains the structured array saved in ControlDesk of the patch 1
% linearization test.  patch2 through patch7 are for patches 2 through 7.
%%%
% Author: Gina A Peterson
%%%
function [K] = myDataAve4(patch1,patch2,patch3,patch4,patch5,patch6,patch7)

sfile(1) = patch1;
sfile(2) = patch2;
sfile(3) = patch3;
sfile(4) = patch4;
sfile(5) = patch5;
sfile(6) = patch6;
sfile(7) = patch7;

% Initialize
% 49 Steps total, 42 Coefficients, 7 patches
ZStotal=zeros(7,49);
for n=1:7
    patch=n;
    % Initialize
    zs = zeros(42,11000); % 1100 data points from each test = 975*1.1218 sec
    zss=zeros(42,49); % Average zern. coef. at each voltage step
    zsss=zeros(42,49); % Delta Zern. Coef between each step
    for k=1:42
        zs(k,:)=sfile(1,n).Y(1,k).Data(1:11000);
        static = sum(zs(k,50:150))/100;
        for i=1:49
            % Take each step to be between n*(0:200)*1.115, i.e. every 20
            % seconds
            p = floor((-175+i*200));%*1.1218);
            zss(k,i)= sum(zs(k,p:p+150))/150 - static;
            if ((i~=1) & (i<=13))
```

```
                zsss(k,i-1)=zss(k,i)-zss(k,i-1);
            elseif ((i~=1) & (13<i<38))
                zsss(k,i-1)=zss(k,i-1)-zss(k,i);
            elseif ((i~=1) & (i>=38))
                zsss(k,i-1)=zss(k,i)-zss(k,i-1);
            end
        end
        zmean(n,k)= mean(zsss(k,1:48));
        zstd(n,k) = std(zsss(k,1:48));
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     % Use this plot to see average change in zernike values
%     figure(n);
%     plot(1:42,zmean,1:42,zmean+zstd,'--',1:42,zmean-zstd,'--')
%     title(['Mean Delta & Standard Deviation for Patch ',int2str(patch)]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Plot Total Surface Change vs voltage for hysteresis
    for g = 1:49
        for i = 1:42
           ZStotal(n,g) = ZStotal(n,g) +(zss(i,g))^2;
        end
    end
    ZStotal = ZStotal.^0.5;
  figure(n);
  plot(-600:50:0,ZStotal(n,37:49),'*k:',600:-50:-600,ZStotal(n,13:37),...
    'xk:',0:50:600,ZStotal(n,1:13),'*k:');
  ylabel('Surface Deflection (microns)'); xlabel('Input Voltage (V)');
  title(['Total Surface Deflection for patch ',int2str(patch),...
    ' over -600v to 600v']);axis tight;
  legend('Positive Voltage Steps','Negative Voltage Steps',0);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate K
A=zmean';
A=A./50; % Divide for 50v steps
% A is in delta microns/1 volt
K=inv(A'*A)*A'; % Volts/Micron Change
K=K./200; % This accounts for amplifier gain
save('KValue', 'K');
```

B.2 SurfacePlot1

This code is used to plot a movie with specified number of frames with the front and overhead view with a bar chart of zernikes at the bottom.

```
function Surface_Plot1(ze,frames)
%%%%%%%%
% Surface_Plot1.m
% Author: Eric M. Trad
%%%
% R : Radial cuts (inside clear aperture)
% P : Azimuthal cuts
% radius : radius of membrane
myfontsize = 24;
radius = 2.5*.0254; %in2m
Rfem = 50;
R = 30; %finite element code is 50 to boundary
r_sf = R/Rfem*radius;
P = 72;
radius_area = 1.0;
[x,y,Zpoly] = Wavefront_Zernikes(R,P,radius_area);
% Initialize surface
for j=1:frames,
    Zsurf(:,:,j)=Zpoly(:,:,1)*0;
    for i=1:42,
        Zsurf(:,:,j)=Zsurf(:,:,j)+ze(j,1,i)*Zpoly(:,:,i);
    end
end
load ZernBasis_512_512_Wavescope
Zpoly1=permute(Zpoly,[3 12]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r1 = 128;
r2 = 0;
si=512;
mi = floor(si/2);
mi=mi+1;
aperture = zeros(si,si);
for i = 1:si
for j = 1:si
dist = sqrt((i-mi)^2+(j-mi)^2); %radial distance
if(dist<r1)
if(dist>=r2)
aperture(i,j) = 1;
end
end
end %j
end %i
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for k = 1:frames,
ax1=subplot(2,2,1);
surf(x*r_sf,y*r_sf,Zsurf(:,:,k))
axis square
scale_z = 1.E-5;%%%Change HERE
```

```
axis([-.08 .08 -.08 .08 -scale_z scale_z]);
load mycolormap
caxis([-scale_z scale_z]);
% title(titlestring,'fontsize',myfontsize)
shading interp
colormap(mycolormap);
colorbar('fontsize',10)
view([0 0 1]);
hold on; %added
%%*************************************************************************
% draw the radius of the Wavescope
% nn : number of elements
% tt : theta vector
% rr : radius vector
% zz : z vector (zeros)
% scale_z : location of surface in plane
%*************************************************************************
nn = 72;
radius = 1.5*.0254; %m %r_sf
rr = ones(1,nn+1)*radius;
tt = [0:2*pi/nn:2*pi];
zz = scale_z * ones(1,nn+1);
[xx,yy,zz] = pol2cart(tt,rr,zz);
plot3(xx,yy,zz,'k');
nn = 72;
radius = 2.5*.0254; %m
rr = ones(1,nn+1)*radius;
tt = [0:2*pi/nn:2*pi];
zz = scale_z * ones(1,nn+1);
[xx,yy,zz] = pol2cart(tt,rr,zz);
plot3(xx,yy,zz,'k');
% % draw the patch pattern
%patch 1
nn = 18;
rr = ones(1,nn+1)*0.5*.0254;
tt = [0:2*pi/nn:2*pi];
zz = scale_z * ones(1,nn+1);
[xx,yy,zz] = pol2cart(tt,rr,zz);
plot3(xx,yy,zz,'k.');
%control lead
nn=12;
rr = [0:2.5*.0254/nn:2.5*.0254];
tt = ones(1,nn+1)*pi/2;
zz = scale_z * ones(1,nn+1);
[xx,yy,zz] = pol2cart(tt,rr,zz);
plot3(xx,yy,zz,'k.');
% %Draw r=0.75 inch lines
nn = 24;
rr = ones(1,nn+7)*0.75*.0254;
tt = [0          :deg2rad(50)/(nn/6):deg2rad(25),...
      deg2rad(35):deg2rad(50)/(nn/6) :deg2rad(85),...
      deg2rad(95):deg2rad(50)/(nn/6) :deg2rad(145),...
```

```
      deg2rad(155):deg2rad(50)/(nn/6) :deg2rad(205),...
      deg2rad(215):deg2rad(50)/(nn/6) :deg2rad(265),...
      deg2rad(275):deg2rad(50)/(nn/6) :deg2rad(325),...
      deg2rad(335):deg2rad(50)/(nn/6) :deg2rad(360),];
zz = scale_z * ones(size(rr));
[xx,yy,zz] = pol2cart(tt,rr,zz);
plot3(xx,yy,zz,'k.');
%Draw r=1.75 inch lines
nn = 36;
rr = ones(1,nn+7)*1.75*.0254;
tt = [0          :deg2rad(50)/(nn/6):deg2rad(25),...
      deg2rad(35):deg2rad(50)/(nn/6) :deg2rad(85),...
      deg2rad(95):deg2rad(50)/(nn/6) :deg2rad(145),...
      deg2rad(155):deg2rad(50)/(nn/6) :deg2rad(205),...
      deg2rad(215):deg2rad(50)/(nn/6) :deg2rad(265),...
      deg2rad(275):deg2rad(50)/(nn/6) :deg2rad(325),...
      deg2rad(335):deg2rad(50)/(nn/6) :deg2rad(360),];
zz = scale_z * ones(size(rr));
[xx,yy,zz] = pol2cart(tt,rr,zz);
plot3(xx,yy,zz,'k.');
%Draw Radials
radials = [25 35 85 95 145 155 205 215 265 275 325 335];
nn = 6;
rr = (0.75:(1.75-0.75)/nn:1.75)*.0254;
zz = scale_z * ones(size(rr));
for i = 1:length(radials)
    tt = ones(size(rr))*deg2rad(radials(i));
    [xx,yy,zz] = pol2cart(tt,rr,zz);
        plot3(xx,yy,zz,'k.');
end hold off;
% 3D View
ax2=subplot(2,2,2);
surf(x*r_sf,y*r_sf,Zsurf(:,:,k))
axis square
%scale_z = 5.E-7; %%Change HERE
axis([-.08 .08 -.08 .08 -scale_z scale_z]);
caxis([-scale_z scale_z]);
% title(titlestring,'fontsize',myfontsize)
shading interp load mycolormap colormap(mycolormap);
colorbar('fontsize',10)
% bar chart of changing zernikes
subplot(2,2,3:4)
path(path,'C:\Documents and Settings\Administrator\Desktop\Matlab\Shepherd');
bar(ze(k,:)./1e-6);
drawnow
end
```

B.3 ZernikeData

This code was used to process all test data. Quick surface plots can be made with this code.

```
%%%%%%%%%
% ZernikeData.m
% ze is the surface in zernikes that can be plotted by SurfacePlot
% Author: Gina A. Peterson and Andrew Marcum
%%%
function [ze] = ZernikeData(stat_ref_struct,dyn_struct,frames)
sfile = stat_ref_struct;
dfile = dyn_struct;
scale_factor = 1E-6;
%Static Reference Surface Analysis
k=1;
for i=1:length(sfile.Y(1,2).Data)-1,
    if(sfile.Y(1,1).Data(i) ~= sfile.Y(1,1).Data(i+1))
        Zstat(k,1,1) = sfile.Y(1,1).Data(i);
        Zstat(k,1,2) = sfile.Y(1,2).Data(i);
        Zstat(k,1,3) = sfile.Y(1,3).Data(i);
        Zstat(k,1,4) = sfile.Y(1,4).Data(i);
        Zstat(k,1,5) = sfile.Y(1,5).Data(i);
        Zstat(k,1,6) = sfile.Y(1,6).Data(i);
        Zstat(k,1,7) = sfile.Y(1,7).Data(i);
        Zstat(k,1,8) = sfile.Y(1,8).Data(i);
        Zstat(k,1,9) = sfile.Y(1,9).Data(i);
        Zstat(k,1,10)= sfile.Y(1,10).Data(i);
        Zstat(k,1,11)= sfile.Y(1,11).Data(i);
        Zstat(k,1,12)= sfile.Y(1,12).Data(i);
        Zstat(k,1,13)= sfile.Y(1,13).Data(i);
        Zstat(k,1,14)= sfile.Y(1,14).Data(i);
        Zstat(k,1,15)= sfile.Y(1,15).Data(i);
        Zstat(k,1,16)= sfile.Y(1,16).Data(i);
        Zstat(k,1,17)= sfile.Y(1,17).Data(i);
        Zstat(k,1,18)= sfile.Y(1,18).Data(i);
        Zstat(k,1,19)= sfile.Y(1,19).Data(i);
        Zstat(k,1,20)= sfile.Y(1,20).Data(i);
        Zstat(k,1,21)= sfile.Y(1,21).Data(i);
        Zstat(k,1,22)= sfile.Y(1,22).Data(i);
        Zstat(k,1,23)= sfile.Y(1,23).Data(i);
        Zstat(k,1,24)= sfile.Y(1,24).Data(i);
        Zstat(k,1,25)= sfile.Y(1,25).Data(i);
        Zstat(k,1,26)= sfile.Y(1,26).Data(i);
        Zstat(k,1,27)= sfile.Y(1,27).Data(i);
        Zstat(k,1,28)= sfile.Y(1,28).Data(i);
        Zstat(k,1,29)= sfile.Y(1,29).Data(i);
        Zstat(k,1,30)= sfile.Y(1,30).Data(i);
        Zstat(k,1,31)= sfile.Y(1,31).Data(i);
        Zstat(k,1,32)= sfile.Y(1,32).Data(i);
        Zstat(k,1,33)= sfile.Y(1,33).Data(i);
        Zstat(k,1,34)= sfile.Y(1,34).Data(i);
        Zstat(k,1,35)= sfile.Y(1,35).Data(i);
        Zstat(k,1,36)= sfile.Y(1,36).Data(i);
```

```
        Zstat(k,1,37)= sfile.Y(1,37).Data(i);
        Zstat(k,1,38)= sfile.Y(1,38).Data(i);
        Zstat(k,1,39)= sfile.Y(1,39).Data(i);
        Zstat(k,1,40)= sfile.Y(1,40).Data(i);
        Zstat(k,1,41)= sfile.Y(1,41).Data(i);
        Zstat(k,1,42)= sfile.Y(1,42).Data(i);
        k=k+1;
   end
end
ZAstat = mean(Zstat);
%Dynamic Surface Analysis
k=1;
for i=1:length(dfile.Y(1,2).Data)-1,
    if(dfile.Y(1,1).Data(i) ~= dfile.Y(1,1).Data(i+1))
        Zdyn(k,1,1) = dfile.Y(1,1).Data(i);
        Zdyn(k,1,2) = dfile.Y(1,2).Data(i);
        Zdyn(k,1,3) = dfile.Y(1,3).Data(i);
        Zdyn(k,1,4) = dfile.Y(1,4).Data(i);
        Zdyn(k,1,5) = dfile.Y(1,5).Data(i);
        Zdyn(k,1,6) = dfile.Y(1,6).Data(i);
        Zdyn(k,1,7) = dfile.Y(1,7).Data(i);
        Zdyn(k,1,8) = dfile.Y(1,8).Data(i);
        Zdyn(k,1,9) = dfile.Y(1,9).Data(i);
        Zdyn(k,1,10)= dfile.Y(1,10).Data(i);
        Zdyn(k,1,11)= dfile.Y(1,11).Data(i);
        Zdyn(k,1,12)= dfile.Y(1,12).Data(i);
        Zdyn(k,1,13)= dfile.Y(1,13).Data(i);
        Zdyn(k,1,14)= dfile.Y(1,14).Data(i);
        Zdyn(k,1,15)= dfile.Y(1,15).Data(i);
        Zdyn(k,1,16)= dfile.Y(1,16).Data(i);
        Zdyn(k,1,17)= dfile.Y(1,17).Data(i);
        Zdyn(k,1,18)= dfile.Y(1,18).Data(i);
        Zdyn(k,1,19)= dfile.Y(1,19).Data(i);
        Zdyn(k,1,20)= dfile.Y(1,20).Data(i);
        Zdyn(k,1,21)= dfile.Y(1,21).Data(i);
        Zdyn(k,1,22)= dfile.Y(1,22).Data(i);
        Zdyn(k,1,23)= dfile.Y(1,23).Data(i);
        Zdyn(k,1,24)= dfile.Y(1,24).Data(i);
        Zdyn(k,1,25)= dfile.Y(1,25).Data(i);
        Zdyn(k,1,26)= dfile.Y(1,26).Data(i);
        Zdyn(k,1,27)= dfile.Y(1,27).Data(i);
        Zdyn(k,1,28)= dfile.Y(1,28).Data(i);
        Zdyn(k,1,29)= dfile.Y(1,29).Data(i);
        Zdyn(k,1,30)= dfile.Y(1,30).Data(i);
        Zdyn(k,1,31)= dfile.Y(1,31).Data(i);
        Zdyn(k,1,32)= dfile.Y(1,32).Data(i);
        Zdyn(k,1,33)= dfile.Y(1,33).Data(i);
        Zdyn(k,1,34)= dfile.Y(1,34).Data(i);
        Zdyn(k,1,35)= dfile.Y(1,35).Data(i);
        Zdyn(k,1,36)= dfile.Y(1,36).Data(i);
        Zdyn(k,1,37)= dfile.Y(1,37).Data(i);
        Zdyn(k,1,38)= dfile.Y(1,38).Data(i);
```

```
        Zdyn(k,1,39)= dfile.Y(1,39).Data(i);
        Zdyn(k,1,40)= dfile.Y(1,40).Data(i);
        Zdyn(k,1,41)= dfile.Y(1,41).Data(i);
        Zdyn(k,1,42)= dfile.Y(1,42).Data(i);
        k=k+1;
    end
end
start=floor(length(Zdyn(:,1,1))/frames);%
k=1;
   for i=1:frames,
     ZAdyn(k,1,:) = mean(Zdyn(i*start-start+1:i*start,1:42),1);
     k=k+1;
   end
za=Zdyn;
Z = zeros(frames,1,42);
% Subtract static frame
 for i=1:frames;
     Z(i,1,:) = ZAdyn(i,1,:)-ZAstat;
 end
ze = Z.*scale_factor;
figure (2); Surface_Plot1(ze,frames);
```

B.4 GetZe0

This code is used to calculate Ze0 for all tests from a saved static array.

```
%%%%%%%%
%GetZe0.m
% Author: Gina A. Peterson
%%%

sfile = stat_ref_struct;
scale_factor = 1E-6;

%Static Reference Surface Analysis
k=1;
for i=1:length(sfile.Y(1,2).Data)-1,
    if(sfile.Y(1,1).Data(i) ~= sfile.Y(1,1).Data(i+1))
        Zstat(k,1,1) = sfile.Y(1,1).Data(i);
        Zstat(k,1,2) = sfile.Y(1,2).Data(i);
        Zstat(k,1,3) = sfile.Y(1,3).Data(i);
        Zstat(k,1,4) = sfile.Y(1,4).Data(i);
        Zstat(k,1,5) = sfile.Y(1,5).Data(i);
        Zstat(k,1,6) = sfile.Y(1,6).Data(i);
        Zstat(k,1,7) = sfile.Y(1,7).Data(i);
        Zstat(k,1,8) = sfile.Y(1,8).Data(i);
        Zstat(k,1,9) = sfile.Y(1,9).Data(i);
        Zstat(k,1,10)= sfile.Y(1,10).Data(i);
        Zstat(k,1,11)= sfile.Y(1,11).Data(i);
        Zstat(k,1,12)= sfile.Y(1,12).Data(i);
        Zstat(k,1,13)= sfile.Y(1,13).Data(i);
        Zstat(k,1,14)= sfile.Y(1,14).Data(i);
        Zstat(k,1,15)= sfile.Y(1,15).Data(i);
        Zstat(k,1,16)= sfile.Y(1,16).Data(i);
        Zstat(k,1,17)= sfile.Y(1,17).Data(i);
        Zstat(k,1,18)= sfile.Y(1,18).Data(i);
        Zstat(k,1,19)= sfile.Y(1,19).Data(i);
        Zstat(k,1,20)= sfile.Y(1,20).Data(i);
        Zstat(k,1,21)= sfile.Y(1,21).Data(i);
        Zstat(k,1,22)= sfile.Y(1,22).Data(i);
        Zstat(k,1,23)= sfile.Y(1,23).Data(i);
        Zstat(k,1,24)= sfile.Y(1,24).Data(i);
        Zstat(k,1,25)= sfile.Y(1,25).Data(i);
        Zstat(k,1,26)= sfile.Y(1,26).Data(i);
        Zstat(k,1,27)= sfile.Y(1,27).Data(i);
        Zstat(k,1,28)= sfile.Y(1,28).Data(i);
        Zstat(k,1,29)= sfile.Y(1,29).Data(i);
        Zstat(k,1,30)= sfile.Y(1,30).Data(i);
        Zstat(k,1,31)= sfile.Y(1,31).Data(i);
        Zstat(k,1,32)= sfile.Y(1,32).Data(i);
        Zstat(k,1,33)= sfile.Y(1,33).Data(i);
        Zstat(k,1,34)= sfile.Y(1,34).Data(i);
        Zstat(k,1,35)= sfile.Y(1,35).Data(i);
        Zstat(k,1,36)= sfile.Y(1,36).Data(i);
        Zstat(k,1,37)= sfile.Y(1,37).Data(i);
```

```
        Zstat(k,1,38)= sfile.Y(1,38).Data(i);
        Zstat(k,1,39)= sfile.Y(1,39).Data(i);
        Zstat(k,1,40)= sfile.Y(1,40).Data(i);
        Zstat(k,1,41)= sfile.Y(1,41).Data(i);
        Zstat(k,1,42)= sfile.Y(1,42).Data(i);
        k=k+1;
    end
end

za = mean(Zstat);
Ze_0 = za(1,:);

% Plot static surface
ze = za.*scale_factor;
figure (1); Surface_Plot1(ze,1);
```

B.5 WavefrontZernikes

This code generates the Zernike Polynomials for use with the acquisition code.

```
function[x,y,Zpoly] = Wavefront_Zernikes(R,P,radius_area);

%**********************************************************************
% PURPOSE: The Wavefront_Zernikes function generates a set of basis elements
% for plotting the Zernike polynomials from the Zernike Coefficients
% generated by the Wavefront sensor. The basis elements are consistent with
% only the Wavefront software, as the ordering of the Zernike polynomials
% by Wavefront is non-standard.
%
% Inputs:
% R : number of radial cuts
% P : number of angular cuts
% radius_area: radius of cicular area (Optional)
% The basis elements, Zpoly, may be plotted by using the surf command, i.e.
% surf(x,y,Zpoly(:,:,4) prints the fourth polynomial.
%
% **** OR *****
% single input R
% R : [n x 3] array of coordinates such that
% R(1,:) is the index
% R(2,:) is the radial coordinate
% R(3,:) is the theta coordinate
%
% The basis elements, Zpoly, may be plotted by using the plot3 command,
% i.e.
% plot3(x,y,Zpoly(:,:,4),'.') prints the fourth polynomial
%

%
%  WRITTEN  BY:  Maj Shepherd
%                Air Force Institute of Technology
%                Aeronautics and Astronautics Department
%                AFIT/ENY
%                2950 P Street
%                Wright-Patterson AFB, OH 45433-7765
%
%  CREATED:  18 Jan 05
%  REVISED:  n/a
%
%**********************************************************************

%**********************************************************************

if nargin > 1
if nargin == 2
radius_area = 1;
end;
radius = [0:radius_area/R:radius_area];
theta = [-pi:(2*pi)/P:pi];
```

```
for i = 1:R+1;
for j = 1:P+1;
    r = radius(i);
    t = theta(j);
x(i,j) = r*cos(t);
y(i,j) = r*sin(t);
Zpoly(i,j,1)  = r*cos(t); %(X Tilt)
Zpoly(i,j,2)  = r*sin(t); % (Y Tilt)
Zpoly(i,j,3)  = 2*r^2-1;  %(Focus)
Zpoly(i,j,4)  = r^2*cos(2*t); %(0 Astigmatism)
Zpoly(i,j,5)  = r^2*sin(2*t); %(45 Astigmatism)
Zpoly(i,j,6)  = (3*r^2-2)*r*cos(t); %(X Coma)
Zpoly(i,j,7)  = (3*r^2-2)*r*sin(t); %(Y Coma)
Zpoly(i,j,8)  = 6*r^4-6*r^2+1; %(Spherical)
Zpoly(i,j,9)  = r^3*cos(3*t);
Zpoly(i,j,10) = r^3*sin(3*t);
Zpoly(i,j,11) = (4*r^2-3)*r^2*cos(2*t);
Zpoly(i,j,12) = (4*r^2-3)*r^2*sin(2*t);
Zpoly(i,j,13) = (10*r^4-12*r^2+3)*r*cos(t);
Zpoly(i,j,14) = (10*r^4-12*r^2+3)*r*sin(t);
Zpoly(i,j,15) = 20*r^6-30*r^4+12*r^2-1;
Zpoly(i,j,16) = r^4*cos(4*t);
Zpoly(i,j,17) = r^4*sin(4*t);
Zpoly(i,j,18) = (5*r^2-4)*r^3*cos(3*t);
Zpoly(i,j,19) = (5*r^2-4)*r^3*sin(3*t);
Zpoly(i,j,20) = (15*r^4-20*r^2+6)*r^2*cos(2*t);
Zpoly(i,j,21) = (15*r^4-20*r^2+6)*r^2*sin(2*t);
Zpoly(i,j,22) = (35*r^6-60*r^4+30*r^2-4)*r*cos(t);
Zpoly(i,j,23) = (35*r^6-60*r^4+30*r^2-4)*r*sin(t);
Zpoly(i,j,24) = 70*r^8-140*r^6+90*r^4-20*r^2+1;
Zpoly(i,j,25) = r^5*cos(5*t);
Zpoly(i,j,26) = r^5*sin(5*t);
Zpoly(i,j,27) = (6*r^2-5)*r^4*cos(4*t);
Zpoly(i,j,28) = (6*r^2-5)*r^4*sin(4*t);
Zpoly(i,j,29) = (21*r^4-30*r^2+10)*r^3*cos(3*t);
Zpoly(i,j,30) = (21*r^4-30*r^2+10)*r^3*sin(3*t);
Zpoly(i,j,31) = (56*r^6-105*r^4+60*r^2-10)*r^2*cos(2*t);
Zpoly(i,j,32) = (56*r^6-105*r^4+60*r^2-10)*r^2*sin(2*t);
Zpoly(i,j,33) = (126*r^8-280*r^6+210*r^4-60*r^2+5)*r*cos(t);
Zpoly(i,j,34) = (126*r^8-280*r^6+210*r^4-60*r^2+5)*r*sin(t);
Zpoly(i,j,35) = 252*r^10-630*r^8+560*r^6-210*r^4+30*r^2-1;
Zpoly(i,j,36) = r^6*cos(6*t);
Zpoly(i,j,37) = r^6*sin(6*t);
Zpoly(i,j,38) = (7*r^2-6)*r^5*cos(5*t);
Zpoly(i,j,39) = (7*r^2-6)*r^5*sin(5*t);
Zpoly(i,j,40) = 924*r^12-2772*r^10+3150*r^8-1680*r^6+420*r^4-42*r^2+1;
Zpoly(i,j,41) = r^7*cos(7*t);
Zpoly(i,j,42) = r^7*sin(7*t);
end %i
end %j
else
%     R = 2 %test line
```

```
%     P = 8 %test line
% [X1X2,EL] = FEM_Membrane_cylindrical(R,P); %test line
X1X2 = R;
for i = 1:length(X1X2(:,1));
    r = X1X2(i,2);
    t = X1X2(i,3);
    x(i,1) = r*cos(t);
    y(i,1) = r*sin(t);
Zpoly(i,1,1)  = r*cos(t); %(X Tilt)
Zpoly(i,1,2)  = r*sin(t); % (Y Tilt)
Zpoly(i,1,3)  = 2*r^2-1;
Zpoly(i,1,4)  = r^2*cos(2*t); %(0 Astigmatism)
Zpoly(i,1,5)  = r^2*sin(2*t); %(45 Astigmatism)
Zpoly(i,1,6)  = (3*r^2-2)*r*cos(t); %(X Coma)
Zpoly(i,1,7)  = (3*r^2-2)*r*sin(t); %(Y Coma)
Zpoly(i,1,8)  = 6*r^4-6*r^2+1; %(Spherical)
Zpoly(i,1,9)  = r^3*cos(3*t);
Zpoly(i,1,10) = r^3*sin(3*t);
Zpoly(i,1,11) = (4*r^2-3)*r^2*cos(2*t);
Zpoly(i,1,12) = (4*r^2-3)*r^2*sin(2*t);
Zpoly(i,1,13) = (10*r^4-12*r^2+3)*r*cos(t);
Zpoly(i,1,14) = (10*r^4-12*r^2+3)*r*sin(t);
Zpoly(i,1,15) = 20*r^6-30*r^4+12*r^2-1;
Zpoly(i,1,16) = r^4*cos(4*t);
Zpoly(i,1,17) = r^4*sin(4*t);
Zpoly(i,1,18) = (5*r^2-4)*r^3*cos(3*t);
Zpoly(i,1,19) = (5*r^2-4)*r^3*sin(3*t);
Zpoly(i,1,20) = (15*r^4-20*r^2+6)*r^2*cos(2*t);
Zpoly(i,1,21) = (15*r^4-20*r^2+6)*r^2*sin(2*t);
Zpoly(i,1,22) = (35*r^6-60*r^4+30*r^2-4)*r*cos(t);
Zpoly(i,1,23) = (35*r^6-60*r^4+30*r^2-4)*r*sin(t);
Zpoly(i,1,24) = 70*r^8-140*r^6+90*r^4-20*r^2+1;
Zpoly(i,1,25) = r^5*cos(5*t);
Zpoly(i,1,26) = r^5*sin(5*t);
Zpoly(i,1,27) = (6*r^2-5)*r^4*cos(4*t);
Zpoly(i,1,28) = (6*r^2-5)*r^4*sin(4*t);
Zpoly(i,1,29) = (21*r^4-30*r^2+10)*r^3*cos(3*t);
Zpoly(i,1,30) = (21*r^4-30*r^2+10)*r^3*sin(3*t);
Zpoly(i,1,31) = (56*r^6-105*r^4+60*r^2-10)*r^2*cos(2*t);
Zpoly(i,1,32) = (56*r^6-105*r^4+60*r^2-10)*r^2*sin(2*t);
Zpoly(i,1,33) = (126*r^8-280*r^6+210*r^4-60*r^2+5)*r*cos(t);
Zpoly(i,1,34) = (126*r^8-280*r^6+210*r^4-60*r^2+5)*r*sin(t);
Zpoly(i,1,35) = 252*r^10-630*r^8+560*r^6-210*r^4+30*r^2-1;
Zpoly(i,1,36) = r^6*cos(6*t);
Zpoly(i,1,37) = r^6*sin(6*t);
Zpoly(i,1,38) = (7*r^2-6)*r^5*cos(5*t);
Zpoly(i,1,39) = (7*r^2-6)*r^5*sin(5*t);
Zpoly(i,1,40) = 924*r^12-2772*r^10+3150*r^8-1680*r^6+420*r^4-42*r^2+1;
Zpoly(i,1,41) = r^7*cos(7*t);
Zpoly(i,1,42) = r^7*sin(7*t);
end; % i loop
end; %nargin if then
```

B.6 8-Byte Model Binary Convert Code

Code is in the Converting MATLAB Function blocks in the 8-byte Model. They are embedded MATLAB blocks.

```
%%%%%%%%
%Convertcode.m
%%%
% Author: Gina A. Peterson
%%%%%%%%
function y = Convert(u)
% Grab Inputs
y1 = dec2bin(u(1),8);
y2 = dec2bin(u(2),8);
y3 = dec2bin(u(3),8);
y4 = dec2bin(u(4),8);

input = [y4 y3 y2 y1]; % This puts bits in 31 to 0 order
a = input(10:32); %This grabs bits 22 to 0
b = input(2:9);  % This grabs bits 30 to 23
c = input(1); % This grabs bit 31, the sign bit

% Need to convert binary to decimal with 2^x
p=23; A = a(1:23); v = A - '0';
Six = [2^22 2^21 2^20 2^19 2^18 2^17 2^16 2^15 2^14 2^13];
SixTwo = [2^12 2^11 2^10 2^9 2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0];
sixes=[Six SixTwo];S=sum(v .* sixes,2);
if (a(23) == '0')
 p=22;A1=a(1:22);v1 = A1 - '0';
 SixOne = [2^21 2^20 2^19 2^18 2^17 2^16 2^15 2^14 2^13];
 sixes1=[SixOne SixTwo];S=sum(v1 .* sixes1,2);
 if (a(22) == '0')
  p=21;A2=a(1:21); v2 = A2 - '0';
  SixThree = [2^20 2^19 2^18 2^17 2^16 2^15 2^14 2^13];
  sixes2=[SixThree SixTwo];S=sum(v2 .* sixes2,2);
  if (a(21) == '0')
   p=20; A3=a(1:20); v3 = A3 - '0';
   Sixfour = [2^19 2^18 2^17 2^16 2^15 2^14 2^13];
   sixes3=[Sixfour SixTwo];S=sum(v3 .* sixes3,2);
   if (a(20)== '0')
    p=19; A4 = a(1:19); v4 = A4 - '0';
    Sixfive = [2^18 2^17 2^16 2^15 2^14 2^13];
    sixes4=[Sixfive SixTwo];S=sum(v4 .* sixes4,2);
    if (a(19) == '0')
     p=18; A5=a(1:18);v5 = A5 - '0';
     Sixsix = [2^17 2^16 2^15 2^14 2^13];
     sixes5=[Sixsix SixTwo];S=sum(v5 .* sixes5,2);
     if (a(18) == '0')
      p=17; A6=a(1:17);v6 = A6 - '0';
      Sixseven = [2^16 2^15 2^14 2^13];
      sixes6=[Sixseven SixTwo];S=sum(v6 .* sixes6,2);
      if (a(17) == '0')
       p=16; A7=a(1:16);v7 = A7 - '0';
```

```
Sixeight = [2^15 2^14 2^13];
sixes7=[Sixeight SixTwo];S=sum(v7 .* sixes7,2);
if (a(16) == '0')
 p=15; A8=a(1:15);v8 = A8 - '0';
 Sixnine = [2^14 2^13];
 sixes8=[Sixnine SixTwo];S=sum(v8 .* sixes8,2);
 if (a(15) == '0')
  p=14; A9=a(1:14);v9 = A9 - '0';
  Sixten = [2^13];
  sixes9=[Sixten SixTwo];S=sum(v9 .* sixes9,2);
  if (a(14) == '0')
   p=13; A10=a(1:13);v10 = A10 - '0';
   sixes10=[SixTwo];S=sum(v10 .* sixes10,2);
   if (a(13) == '0')
    p=12; A11=a(1:12);v11 = A11 - '0';
    sixes11=[2^11 2^10 2^9 2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0];
    S=sum(v11 .* sixes11,2);
    if (a(12) == '0')
     p=11; A12=a(1:11);v12 = A12 - '0';
     sixes12=[2^10 2^9 2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0];
     S=sum(v12 .* sixes12,2);
     if (a(11) == '0')
      p=10; A13=a(1:10);v13 = A13 - '0';
      sixes13=[2^9 2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0];
      S=sum(v13 .* sixes13,2);
      if (a(10) == '0')
       p=9; A14=a(1:9);v14 = A14 - '0';
       sixes14=[2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0];
       S=sum(v14 .* sixes14,2);
       if (a(9) == '0')
        p=8; A15=a(1:8);v15 = A15 - '0';
        sixes15=[2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0];
        S=sum(v15 .* sixes15,2);
        if (a(8) == '0')
         p=7; A16=a(1:7);v16 = A16 - '0';
         sixes16=[2^6 2^5 2^4 2^3 2^2 2^1 2^0];
         S=sum(v16 .* sixes16,2);
         if (a(7) == '0')
          p=6; A17=a(1:6);v17 = A17 - '0';
          sixes17=[2^5 2^4 2^3 2^2 2^1 2^0];
          S=sum(v17 .* sixes17,2);
          if (a(6) == '0')
           p=5; A18=a(1:5);v18 = A18 - '0';
           sixes18=[16 8 4 2 1];
           S=sum(v18 .* sixes18,2);
           if (a(5) == '0')
            p=4; A19=a(1:4);v19 = A19 - '0';
            sixes19=[8 4 2 1];
            S=sum(v19 .* sixes19,2);
            if (a(4) == '0')
             p=3; A20=a(1:3);v20 = A20 - '0';
             sixes20=[4 2 1];
```

```
                    S=sum(v20 .* sixes20,2);
                    if (a(3) == '0')
                     p=2; A21=a(1:2);v21 = A21 - '0';
                     sixes21=[2 1];
                     S=sum(v21 .* sixes21,2);
                     if (a(2) == '0')
                      p=1; A22=a(1);v22 = A22 - '0';
                      S=v22;
                      if (a(1) == '0')
                          p=0; S=0;
end;end;end;end;end;end;end;end;end;end;end;end;end;end;end;end;end;
end;end;end;end;end;end;

E = p; % Order of exponent
D=S/2^E; % Significand decimal value
DV=1+D;
B=bin2dec(b); % Exponent Field
BV=B-127;
y=DV*2^BV;
% Derivation from http://babbage.cs.qc.edu/courses/cs341/IEEE-754.html

% Check sign bit
if (c == '1')
    y = -1*y;
end;

% All functions must be hard coded to build model
% This dec2bin and bin2dec code assumes an 8 bit converstion
function s=dec2bin(d,n)
s=char(rem(floor(d*[.0078125 .015625 .03125 .0625 .125 .25 .5 1]),2)+'0');

function x=bin2dec(s)
vv = s - '0'; % assumes 8 inputs
mult = [2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0];
x = sum(vv .* mult,2);
```

B.7 WavefrontZernikes

This code normalizes the Zernike coefficients for use with the WaveScope.

```
function [Cn] = Zernike_Wavescope_Input(Cw)
%*********************************************************************
%  FUNCTIONS: Zernike_Wavescope_Input(Cw);
%
%  PURPOSE: The Zernike_Wavescope_Output function generates a set
%  of scaled Zernike coefficients for input into or comparison against the
%  Wavescope software.
%
%  The normalization is such that:
%
%                       /2pi /1
%                       |    |
%                       |    | Z_i Z_j rdrd@ = d_ij
%                       |    |
%                      0/   0/
%
%                      with normalization constants
%
%                      A_n^m = sqrt(2(n+1)/pi)  m ~= 0
%                      A_n^m = sqrt((n+1)/pi)   m == 0
%
%                      where n is radial degree
%                      and   m is azimuthal frequency
%
%  EXAMPLE: The first Zernike coefficient on the Xtilt mode (Wavescope
%  Zernike #1) is 8.8623e-007. The scaling is 2*1/sqrt(pi) = 1.128. The
%  scaled Zernike coefficient is 8.8623e-007*2*1/sqrt(pi)= 1E-6;
%
%
%  VARIABLES:
%  Cw : Input vector of up to 42 coefficients ordered using Wavescope index
%  Cn : Normalized coeffients (for computations using orthonormal Zernikes)
%
%
%  WRITTEN  BY:  Maj Shepherd
%                Air Force Institute of Technology
%                Aeronautics and Astronautics Department
%                AFIT/ENY
%                2950 P Street
%                Wright-Patterson AFB, OH 45433-7765
%
%  CREATED:  30 Aug 05
%  REVISED:  N/A
if length(Cw) > 42
    disp('Wavefront order is limited to 42 polynomials')
    return
end V = [2 2 sqrt(3),...
     sqrt(6)  sqrt(6)  sqrt(8)  sqrt(8)  sqrt(5),...
     sqrt(8)  sqrt(8)  sqrt(10) sqrt(10) sqrt(12) sqrt(12) sqrt(7),...
```

```
    sqrt(10) sqrt(10) sqrt(12) sqrt(12) sqrt(14) sqrt(14) 4 4 3,...
    sqrt(12) sqrt(12) sqrt(14) sqrt(14) 4  4     sqrt(18) sqrt(18),...
    sqrt(20) sqrt(20) sqrt(11),...
    sqrt(14) sqrt(14) 4  4  sqrt(13),...
    4 4];
V = V*1/sqrt(pi);
A = diag(V,0);
A = A(1:length(Cw),1:length(Cw));
Cn = A*Cw;
```

B.8 WavefrontZernikes

This code normalizes the Zernike coefficients from the WaveScope.

```
function [Cn] = Zernike_Wavescope_Output(Cw)
%**********************************************************************
%  FUNCTIONS: Zernike_Wavescope_Output(Cw);
%
%  PURPOSE: The Zernike_Wavescope_Output function generates a set
%  of scaled Zernike coefficients from an input vector of coefficients from
%  the Wavescope software.
%
%  The normalization is such that:
%
%                       /2pi /1
%                       |    |
%                       |    | Z_i Z_j rdrd@ = d_ij
%                       |    |
%                      0/   0/
%
%                      with normalization constants
%
%                      A_n^m = sqrt(2(n+1)/pi)  m ~= 0
%                      A_n^m = sqrt((n+1)/pi)   m == 0
%
%                      where n is radial degree
%                      and   m is azimuthal frequency
%
%  EXAMPLE: The first Zernike coefficient from Wavescope is Cw = 1E-6;
%           The normalization for the Xtilt mode (Wavescope Zernike #1) is
%           2*1/sqrt(pi) = 1.1284. The normalized Zernike coefficient is
%           then:
%           1E-6/(2*1/sqrt(pi)) = 8.8623e-007.
%
%  VARIABLES:
%  Cw : Input vector of up to 42 coefficients ordered using Wavescope index
%  Cn : Normalized coeffients (for computations using orthonormal Zernikes)
%
%
%  WRITTEN  BY:  Maj Shepherd
%                Air Force Institute of Technology
%                Aeronautics and Astronautics Department
%                AFIT/ENY
%                2950 P Street
%                Wright-Patterson AFB, OH 45433-7765
%
%  CREATED:  30 Aug 05
%  REVISED:  N/A
if length(Cw) > 42
    disp('Wavefront order is limited to 42 polynomials')
    return
end V = [2 2 sqrt(3),...
     sqrt(6)  sqrt(6)  sqrt(8)  sqrt(8)  sqrt(5),...
```

```
    sqrt(8)  sqrt(8)  sqrt(10) sqrt(10) sqrt(12) sqrt(12) sqrt(7),...
    sqrt(10) sqrt(10) sqrt(12) sqrt(12) sqrt(14) sqrt(14) 4 4 3,...
    sqrt(12) sqrt(12) sqrt(14) sqrt(14) 4  4     sqrt(18) sqrt(18),...
    sqrt(20) sqrt(20) sqrt(11),...
    sqrt(14) sqrt(14) 4  4  sqrt(13),...
    4 4];
V = V*1/sqrt(pi);
A = inv(diag(V,0));  %note the inverse here
A = A(1:length(Cw),1:length(Cw));
Cn = A*Cw;
```

B.9 ControlTests

This code was used to process all test data from the control tests. The residual, obtained surfaces, and time history plots can be made with this code.

```
%%%%%%%%
% ControlTests.m
% Author: Gina A. Peterson
%%%
% This code lays out directions to get K, the control matrix and
% use it to get all data out of the control tests

% After have chosen 7 actuator commands and recorded their 42 Zernike
% coefficients for at least 20 seconds in patchtest, record a static test
% with the mirror at rest. Set best 20 sec by setting it in ZerninkeData.m
 [za zs ze] = ZernikeData(static,patchtest,164);

% Get the average ZeControl surface
Zerns = mean(ze,1); %figure(1);Surface_Plot(Zerns2,1);
for i = 1:42; Ze_Control(i) = Zerns(1,1,i);end;
Ze_Control=Ze_Control*1e6;

% Record the history of the surface deformation in patchtest to plot
 His=zeros(1,501);
    for i=1:501;
      for a=1:42;
        His(i) = His(i) + (patchtest.Y(1,a).Data(i)-zs(a))^2;
      end;
    end;
% Plot with plot(0:.1:50,His);

% To get Open-Loop Actuator Commands
 [K] = myDataAve4(1,2,3,4,5,6,7); % 1 through 7 represent each linearization test
 K=K./200;
 K.*200*Ze_Control' % This gives actuator commands

% Now run calculated Open-Loop commands and record in expecttest
[za zs ze] = ZernikeData(static, expecttest,164);% take from best 20 seconds

% Get the Expected Surface plot
Zernex = mean(ze,1); Surface_Plot(Zernex,1);
save('DataPoints','Zernex','His','Zerns','Ze_Control')

% Get Ze_0 for the model to run feedback tests
 [Ze_0] =GetZe0(static);

% Now run closed-loop test and record data in control
 [za zs ze] = ZernikeData(static, control,164);% take best 20 seconds

 % Get obtained surface and residual
 ZernControl = mean(ze,1);
 % save('ZernControl','ZernControl')
```

```
 Surface_Plot(ZernControl-Zernex,1);
  ylabel('y-dir (m)'); xlabel('x-dir (m)');% save this as residual
 Surface_Plot(ZernControl,1);
  ylabel('y-dir (m)'); xlabel('x-dir (m)');%save this as control

% to plot the total surface deformation history, get both time histories of expected
% and control surfaces
 ActiveH=zeros(1,601);
   for i=1:601;
     for a=1:42;
      ActiveH(i) = ActiveH(i) + (control.Y(1,a).Data(i+100)-zs(a))^2;
     end;
   end;
 ActiveH = ActiveH.^.5;
 ActiveH2=zeros(1,201);
   for i=1:201;
     for a=1:42; %grab data after feedback starts, after 10 seconds
       ActiveH2(i) = ActiveH2(i) + (expecttest.Y(1,a).Data(i)-zs(a))^2;
     end;
   end;
 ActiveH2 = ActiveH2.^.5;
 temp=mean(ActiveH2);
 plot(0:.1:60,ActiveH-temp); % This plots Surface Deformation history
 xlabel('Time (seconds)'); ylabel('Total Surface Deflection (microns)');
 title('Total Surface Deflection');

% to plot the residual error history, get both time histories of expected
% and control surfaces
 WW=zeros(1,601);
   for i=1:601;
     for a=1:42;
      WW(i) = WW(i) + (control.Y(1,a).Data(i+100)-zs(a)-Zernsex(a))^2;
     end;
   end;
WW = WW.^.5;
 plot(0:.1:60,WW); % This plots Residual history
 xlabel('Time (seconds)'); ylabel('Residual (microns)');
 title('Residual Error');

% After done with 4 tests and they are numbered sequentially:
% Once get all four Zerns
figure(30);subplot(2,2,1);Surface_Plot(Zerns1,1); title('1');
 ylabel('y-dir (m)'); xlabel('x-dir (m)');
subplot(2,2,2);Surface_Plot(Zerns2,1);title('2');
 ylabel('y-dir(m)'); xlabel('x-dir (m)');
subplot(2,2,3);Surface_Plot(Zerns3,1);title('3');
 ylabel('y-dir(m)'); xlabel('x-dir (m)');
subplot(2,2,4);Surface_Plot(Zerns4,1);title('4');
 ylabel('y-dir(m)'); xlabel('x-dir (m)');

% Once get all four Zernex
figure(34);subplot(2,2,1); Surface_Plot(Zernex1,1); title('1');
```

```
ylabel('y-dir (m)'); xlabel('x-dir (m)');
subplot(2,2,2);Surface_Plot(Zernex2,1);title('2');
ylabel('y-dir(m)'); xlabel('x-dir (m)');
subplot(2,2,3);Surface_Plot(Zernex3,1);title('3');
ylabel('y-dir(m)'); xlabel('x-dir (m)');
subplot(2,2,4);Surface_Plot(Zernex4,1);title('4');
ylabel('y-dir(m)'); xlabel('x-dir (m)');

% Once got all 4 His
% Get a surface variation for the mirror at rest
TimeHis=zeros(1,101);
  for i=1:101;
   for a=1:42;
   TimeHis(i) = TimeHis(i) + (static1.Y(1,a).Data(i))^2;
   end;
  end;
TimeHis=TimeHis.^.5; figure(35);
plot(t,TimeHis-mean(TimeHis),'k',t,His1-mean(His1),':k',t,...
 His2-mean(His2),'--k',t,His3-mean(His3),':k.',t,His4-mean(His4),'--k*');
title('Time History of Total Surface Deflection for Mirror')
 ylabel('Deflection (microns)'); xlabel('Time (seconds)');
 legend('Resting Mirror','Control Surface 1','Control Surface2',...
  'Control Surface 3','Control Surface 4'0)

% To plot the Piezo block changes with time, record data in pzt
 ActiveH=zeros(1,401);
   for i=1:401;
     for a=1:42;
     ActiveH(i) = ActiveH(i) + (pzt.Y(1,a).Data(i)-zs(a))^2;
     end;
   end;
ActiveH = ActiveH.^.5;
 plot(0:.1:40,ActiveH)
```

B.10 AlwaysRun

This code was used to initialize all parameters within the conversion model in order to build the model. The matrix A is made up of the normalization factors.

```
%%%%%%%%
% AlwaysRun.m
% Author: Gina A. Peterson and Maj Shepherd
%%%
 A=[2 2 sqrt(3),...
    sqrt(6)  sqrt(6)  sqrt(8)  sqrt(8)  sqrt(5),...
    sqrt(8)  sqrt(8)  sqrt(10) sqrt(10) sqrt(12) sqrt(12) sqrt(7),...
    sqrt(10) sqrt(10) sqrt(12) sqrt(12) sqrt(14) sqrt(14) 4 4 3,...
    sqrt(12) sqrt(12) sqrt(14) sqrt(14) 4  4  sqrt(18) sqrt(18),...
    sqrt(20) sqrt(20) sqrt(11),...
    sqrt(14) sqrt(14) 4  4  sqrt(13) 4 4];
 A=A*1/sqrt(pi);
 A = inv(diag(A,0));
 K=zeros(7,42);
 Ze_0=zeros(1,42);
 Ze_Control = zeros(1,42);
```

Appendix C. C-Code within Data Acquisition Blocks

C.1 C-code in the 1225-Byte Model

This is the c code found in the ASCII Converter block in the 1225-Byte model.

C.1.1 S-Function Builder2.

```
/*converts decimal values to corresponding ASCII characters
Author: Andrew Marcum
20 July 2005
AFIT / ENY
 */
    int Bytes;
    int i;
     Bytes = 1225;
     for (i=0; i<Bytes; i=i+1){
     switch((int)u0[i]){
          case 10:
             y0[i] = 10;
             break;
          case 13:
             y0[i] = 13;
             break;
          case 45:
             y0[i] = 2.5;
             break;
          case 46:
             y0[i] = 3.5;
             break;
          case 47:
             y0[i] = 1.5;
             break;
          case 48:
             y0[i] = 0;
             break;
          case 49:
             y0[i] = 1;
             break;
          case 50:
             y0[i] = 2;
             break;
          case 51:
             y0[i] = 3;
             break;
          case 52:
             y0[i] = 4;
             break;
          case 53:
             y0[i] = 5;
             break;
          case 54:
             y0[i] = 6;
             break;
```

```
          case 55:
             y0[i] = 7;
             break;
          case 56:
             y0[i] = 8;
             break;
          case 57:
             y0[i] = 9;
             break;
          case 60:
             y0[i] = 5.5;
             break;
          case 62:
             y0[i] = 6.5;
             break;
          case 101:
             y0[i] = 4.5;
             break;
           default:
             y0[i] = 0;
             break;
       }
      }
```

C.1.2 S-Function Builder3.

```
/*Sorts and aligns ASCII characters
Author: Andrew Marcum
20 July 2005
AFIT / ENY
 */
   int i;
   int shift;
   for(shift=0; shift<42; shift=shift+1){
     for(i=(19 + shift*29); i<(34 + shift*29); i=i+1){    /*find decimal points*/
      if(u0[i] == 4.5){
        y0[shift] = 0;
             }
          else{
      if(u0[i] == 3.5){   /*decimal point*/
          if(u0[i-2] == 2.5){   /*minus sign*/
        y0[shift] = -1*(u0[i-1] + u0[i+1]*.1 + u0[i+2]*.01 + u0[i+3]*.001 ...
+ u0[i+4]*.0001 + u0[i+5]*.00001 + u0[i+6]*.000001 + u0[i+7]*.0000001 + u0[i+8]*.00000001);
                      }
          else{    /*plus sign*/
        y0[shift] = u0[i-1] + u0[i+1]*.1 + u0[i+2]*.01 + u0[i+3]*.001 + ...
u0[i+4]*.0001 + u0[i+5]*.00001 + u0[i+6]*.000001 + u0[i+7]*.0000001 + u0[i+8]*.00000001;
                }
            }
            }
         }
       }
```

C.2 C-code in the 1-Byte Model

This is the c code found in the ASCII Conversion and the Converter blocks in the 1-Byte model.

C.2.1 ASCII Filter.

```
/* Author: Andrew Marcum*/
          int i;
          if((u0[0] == 60) && (u0[26] == 62) && (u0[27] == 13)){
              for(i=0; i<29; i++){
                     y0[i] = u0[i];
              }
          }
          else{
              for(i=0; i<29; i++){
                     y0[i] = 0;
              }
          }
```

C.2.2 Dec to ASCII.

```
/*converts decimal values to corresponding ASCII characters
  Author: Andrew Marcum
 20 July 2005
 AFIT / ENY
 */
  int Bytes;
  int i;
     Bytes = 29;
   for (i=0; i<Bytes; i=i+1){
     switch((int)u0[i]){
     case 32:
     y0[i] = 0;
     break;
     case 45:
     y0[i] = 2.5; /*negative sign*/
     break;
     case 46:
     y0[i] = 3.5; /*decimal point*/
     break;
     case 48:
     y0[i] = 0;
     break;
     case 49:
     y0[i] = 1;
     break;
     case 50:
     y0[i] = 2;
     break;
     case 51:
     y0[i] = 3;
```

```
    break;
    case 52:
    y0[i] = 4;
    break;
    case 53:
    y0[i] = 5;
    break;
    case 54:
    y0[i] = 6;
    break;
    case 55:
    y0[i] = 7;
    break;
    case 56:
    y0[i] = 8;
    break;
    case 57:
    y0[i] = 9;
    break;
    case 101:
    y0[i] = 4.5; /* e */
    break;
    default:
    y0[i] = 0;
    break;
      }
    }
```

C.2.3 Coeff. Number Organizer.

```
/* Authors: Andrew Marcum and Gina Peterson*/

 if((u0[0] == 0) && (u0[1] == 0) && (u0[2] == 0) && (u0[3] == 0) && (u0[4] == 0) && (u0[5] == 0)){
   if(u0[7] == 3.5){
     y0[0] = u0[6];
       }
   if(u0[8] == 3.5){
     y0[0] = u0[6]*10 + u0[7];
    }
  }
    else{
     y0[0] = 0;
  }
```

C.2.4 Zernike Organizer.

```
/*Sorts and aligns ASCII characters
  Author: Andrew Marcum
 20 July 2005
 AFIT / ENY
 */
    int i;
   for(i=13; i<29; i=i+1){    /*find decimal points*/
```

```
    if(u0[i] == 4.5){
      y0[0] = 0;
      }
    else{
     if(u0[i] == 3.5){   /*decimal point*/
       if(u0[i-2] == 2.5){   /*minus sign*/
  y0[0] = -1*(u0[i-1] + u0[i+1]*.1 + u0[i+2]*.01 + u0[i+3]*.001 + ...
u0[i+4]*.0001 + u0[i+5]*.00001 + u0[i+6]*.000001 + u0[i+7]*.0000001 + u0[i+8]*.00000001);
      }
       else{    /*plus sign*/

   y0[0] = u0[i-1] + u0[i+1]*.1 + u0[i+2]*.01 + u0[i+3]*.001 + u0[i+4]*.0001 +...
 u0[i+5]*.00001 + u0[i+6]*.000001 + u0[i+7]*.0000001 + u0[i+8]*.00000001;
       }
     }
    }
  }
```

C.2.5 Zernike Matrix Builder.

```
/* Authors: Gina Peterson and Andrew Marcum*/

                              if(u0[0]==1){
                                  y0[0] = u1[0];}
                              if(u0[0]==2){
                                  y1[0] = u1[0]; }
                              if(u0[0]==3){
                                  y2[0] = u1[0]; }
                              if(u0[0]==4){
                                  y3[0] = u1[0]; }
                              if(u0[0]==5){
                                  y4[0] = u1[0]; }
                              if(u0[0]==6){
                                  y5[0] = u1[0]; }
                              if(u0[0]==7){
                                  y6[0] = u1[0]; }
                              if(u0[0]==8){
                                  y7[0] = u1[0]; }
                              if(u0[0]==9){
                                  y8[0] = u1[0]; }
                              if(u0[0]==10){
                                  y9[0] = u1[0]; }
                              if(u0[0]==11){
                                  y10[0] = u1[0]; }
                              if(u0[0]==12){
                                  y11[0] = u1[0]; }
                              if(u0[0]==13){
                                  y12[0] = u1[0]; }
                              if(u0[0]==14){
                                  y13[0] = u1[0]; }
                              if(u0[0]==15){
                                  y14[0] = u1[0]; }
                              if(u0[0]==16){
```

```
    y15[0] = u1[0]; }
if(u0[0]==17){
    y16[0] = u1[0]; }
if(u0[0]==18){
    y17[0] = u1[0]; }
if(u0[0]==19){
    y18[0] = u1[0]; }
if(u0[0]==20){
    y19[0] = u1[0]; }
if(u0[0]==21){
    y20[0] = u1[0];}
if(u0[0]==22){
    y21[0] = u1[0]; }
if(u0[0]==23){
    y22[0] = u1[0]; }
if(u0[0]==24){
    y23[0] = u1[0];  }
if(u0[0]==25){
    y24[0] = u1[0]; }
if(u0[0]==26){
    y25[0] = u1[0]; }
if(u0[0]==27){
    y26[0] = u1[0]; }
if(u0[0]==28){
    y27[0] = u1[0];}
if(u0[0]==29){
    y28[0] = u1[0]; }
if(u0[0]==30){
    y29[0] = u1[0]; }
if(u0[0]==31){
    y30[0] = u1[0]; }
if(u0[0]==32){
    y31[0] = u1[0];  }
if(u0[0]==33){
    y32[0] = u1[0]; }
```

C.3 C-code in the 8-Byte Model

This is the c-code found in the Converter block in the 8-Byte model.

C.3.1 byte Filter.

```
/* Author: Gina Peterson*/
  int i;
  if((u0[0] == 0) && (u0[1] == 0) && (u0[4] != 0)) {
    for(i=0; i<8; i++){
     y0[i] = u0[i];
      }
   }
  else{
    for(i=0; i<8; i++){
     y0[i] = 0;
      }
   }
```

C.3.2 Matrix Byte. This code is exactly as shown for the *Zernike Matrix Builder* block in the 1-byte model in Appendix C.2.5. It includes the additional code:

```
/*Deletes any outrageous numbers
  Author: Gina Peterson
  11 Jan 2006
  AFIT / ENY
  */
    int i;
      for(i=0; i<42; i=i+1){    /*find big numbers*/
      if(u0[i] > 10){
          y0[i] = 0;
   }
        if(u0[i] < -10){
          y0[i] = 0;
   }
        else{
         y0[i] = u0[i];
   }
 }
```

Appendix D. Other Interesting Notes

D.1 Relevant Telephone Numbers

Table 12: Phone Numbers

Name	Phone Number
Adaptive Optics Assoc. (AOA)	617-806-1400
Marc Albanese (AOA)	617-806-1886
Dennis Mansell (AgilOptics)	505-268-4742
Jay Anderson (AFIT/ENY Lab Supervisor)	937-255-3636 x4865
Dr. David Mollenhauer (ML contact)	937-255-9728 (office)
	937-255-9059 (lab)
	937-255-1825 (cell)
Dennis Mansell (Agil Optics President)	505-268-4742

D.2 Zernike Polynomials

Given below are the first 42 Zernike Polynomials. Each term has a coefficient multiplier which scales the set. Upon combination, the surface deflection is given in terms of radius (r) and theta (θ). These coefficients are scaled so that the surface deflection is in terms of microns (μm). The normalization factors (N) for each polynomial is also listed.

$$Z_1 = rcos(\theta),\ N_1 = 2$$

$$Z_2 = rsin(\theta),\ N_2 = 2$$

$$Z_3 = 2r^2 - 1,\ N_3 = \sqrt{3}$$

$$Z_4 = r^2 cos(2\theta),\ N_4 = \sqrt{6}$$

$$Z_5 = r^2 sin(2\theta),\ N_5 = \sqrt{6}$$

$$Z_6 = (3r^2 - 2)rcos(\theta),\ N_6 = \sqrt{8}$$

$$Z_7 = (3r^2 - 2)rsin(\theta),\ N_7 = \sqrt{8}$$

$$Z_8 = 6r^4 - 6r^2 + 1,\ N_8 = \sqrt{5}$$

$$Z_9 = r^3 cos(3\theta),\ N_9 = \sqrt{8}$$

$$Z_{10} = r^3 sin(3\theta),\ N_{10} = \sqrt{8}$$

$$Z_{11} = (4r^2 - 3)r^2 cos(2\theta),\ N_{11} = \sqrt{10}$$

$$Z_{12} = (4r^2 - 3)r^2 sin(2\theta),\ N_{12} = \sqrt{10}$$

$$Z_{13} = (10r^4 - 12r^2 + 3)rcos(\theta),\ N_{13} = \sqrt{12}$$

$$Z_{14} = (10r^4 - 12r^2 + 3)rsin(\theta),\ N_{14} = \sqrt{12}$$

$$Z_{15} = 20r^6 - 30r^4 + 12r^2 - 1,\ N_{15} = \sqrt{7}$$

$$Z_{16} = r^4 cos(4\theta),\ N_{16} = \sqrt{10}$$

$$Z_{17} = r^4 sin(4\theta),\ N_{17} = \sqrt{10}$$

$$Z_{18} = (5r^2 - 4)r^3 cos(3\theta),\ N_{18} = \sqrt{12}$$

$$Z_{19} = (5r^2 - 4)r^3 sin(3\theta),\ N_{19} = \sqrt{12}$$

$$Z_{20} = (15r^4 - 20r^2 + 6)r^2 cos(2\theta),\ N_{20} = \sqrt{14}$$

$$Z_{21} = (15r^4 - 20r^2 + 6)r^2 sin(2\theta),\ N_{21} = \sqrt{14}$$

$$Z_{22} = (35r^6 - 60r^4 + 30r^2 - 4)rcos(\theta),\ N_{22} = 4$$

$$Z_{23} = (35r^6 - 60r^4 + 30r^2 - 4)rsin(\theta),\ N_{23} = 4$$

$$Z_{24} = 70r^8 - 140r^6 + 90r^4 - 20r^2 + 1,\ N_{24} = 3$$

$$Z_{25} = r^5 cos(5\theta),\ N_{25} = \sqrt{12}$$

$$Z_{26} = r^5 sin(5\theta),\ N_{26} = \sqrt{12}$$

$$Z_{27} = (6r^2 - 5)r^4 cos(4\theta),\ N_{27} = \sqrt{14}$$

$$Z_{28} = (6r^2 - 5)r^4 sin(4\theta),\ N_{28} = \sqrt{14}$$

$$Z_{29} = (21r^4 - 30r^2 + 10)r^3 cos(3\theta),\ N_{29} = 4$$

$$Z_{30} = (21r^4 - 30r^2 + 10)r^3 sin(3\theta),\ N_{30} = 4$$

$$Z_{31} = (56r^6 - 105r^4 + 60r^2 - 10)r^2 cos(2\theta),\ N_{31} = \sqrt{18}$$

$$Z_{32} = (56r^6 - 105r^4 + 60r^2 - 10)r^2 sin(2\theta),\ N_{32} = \sqrt{18}$$

$$Z_{33} = (126r^8 - 280r^6 + 210r^4 - 60r^2 + 5)rcos(\theta),\ N_{33} = \sqrt{20}$$

$$Z_{34} = (126r^8 - 280r^6 + 210r^4 - 60r^2 + 5)rsin(\theta),\ N_{34} = \sqrt{18}$$

$$Z_{35} = 252r^10 - 630r^8 + 560r^6 - 210r^4 + 30r^2 - 1,\ N_{35} = \sqrt{11}$$

$$Z_{36} = r^6 cos(6\theta),\ N_{36} = \sqrt{14}$$

$$Z_{37} = r^6 sin(6\theta),\ N_{37} = \sqrt{14}$$

$$Z_{38} = (7r^2 - 6)r^5 cos(5\theta),\ N_{38} = 4$$

$$Z_{39} = (7r^2 - 6)r^5 sin(5\theta),\ N_{39} = 4$$

$$Z_{40} = 924r^12 - 2772r^10 + 3150r^8 - 1680r^6 + 420r^4 - 42r^2 + 1,\ N_{40} = \sqrt{13}$$

$$Z_{41} = r^7 cos(7\theta),\ N_{41} = 4$$

$$Z_{42} = r^7 sin(7\theta),\ N_{42} = 4$$

Bibliography

1. Agnes, G. S. and others, . "Shaping of Parabolic Cylindrical Membrane Reflectors for the DART Precision Test Bed," *American Institute of Aeronautics and Astronautics*, *1661*:1–12 (2004).

2. Angel, R. and others, . "Stretched Membrane with Electrostatic Curvature (SMEC): A New Technology for Ultra-Lightweight Space Telescopes.".

3. Arnold, L. "Uniform-Load and Actuator Influence Functions of a Thin or Thick Annular Mirror: Applications to Active Mirror Support Optimization," *Applied Optics*, *Vol. 35, No. 7*:1095–1106 (March 1996).

4. Ash, J. T. and H., J. C. "Shape Achievement of Optical Membrane Mirrors Using Coating/Substrate Intrinsic Stresses," *Structural, Structural Dynamics, and Materials Conference, Denver, CO*, 1–9 (April 2002).

5. Bao, X. and others, . "Numerical Modeling of Electroactive Polymer Mirrors for Space Applications," *SPIE Smart Structures and Materials Symposium, EAPAD Conference*, *Vol. 5051-45* (2003).

6. Bennet, H. E. and others, . "Development of Lightweight Mirror Elements for the Euro50 Mirrors," *Proceedings of SPIE*, *5382*:526–533 (2004).

7. Bilbro, J. "Optics in Orbit," *Spie's Oemagazine* (2001).

8. Brown, R. H., "Energy Generation Using Piezo Film." Online at http://www.meas-spec.com/myMSI/sensors/FilmSheets.asp, January 2006.

9. Burge, J. H. and others, . "Active Mirror Technology for Large Space Telescopes," *UV, Optical, and IR Space Telescopes and Instruments, Proceedings of SPIE*, *4013*:640–648 (2000).

10. Bush, K. and others, . "Electrostatic Membrane Deformable Mirror Wavefront Control Systems: Design and Analysis," *Advanced Wavefront Control: Methods, Devices, and Applications, SPIE*, *5553* (2004).

11. Clark, G., "Web-based instruction based on audio narritive from on-line SOS course." College of Aerospace Doctrine, Reserch, and Education (CADRE), Maxwell AFB, AL, July 2005.

12. Clark, G., "Space News Website." Online at http://www.space.com/news/gossamer.html, January 2006.

13. Dargaville, T. R. and others, . "Evaluation of Piezoelectric PVDF Polymers for Use in Space Enviroments," *Structures, Structural Dynamics and Materials Conference, Palm Springs, CA*, 1–8 (April 2004).

14. Devilliers, C. and R., K. "CESIC -A New Technology for Lightweight and Cost Effective Space Instrument Structures and Mirrors," *Optical Materials and Structures Technologies II, Proceedings of SPIE*, *5868*:1–18 (2005).

15. Dimakov, S. A. and others, . "Control of Membrane Mirror Profiel by Electrostatic Field," *Proceedings of SPIE*, *Vol. 4091*:137–142 (2000).

16. Dimitry, G. and others, . "Distributed Localized Shape Control of Gossamer Space Structures," *American Institute of Aeronautics and Astronautics*, *1197*:1–8 (2001).

17. Dixit, S. and others, . "Development of Large-Aperture, Light-Weight Fresnel Lenses for Gossamer Space Telescopes," *Structural Dynamics and Materials Conference*, *April*:22–25 (2002).

18. Fernandez, E. J. and Artal, P. "Membrane Deformable Mirror for Adaptive Optics: Performance Limits in Visual Optics," *Optics Express*, *Vol 11 No 9*:1056–1058 (2003).

19. Flint, E. M. and K., D. K. "Approach for Efficiently Evaluating Internally Reacted Global Shape Control Actuation Strategies for Apertures," *44th Structures, Structural Dynamics, and Materials Converence, Norfolk, VA*, *April*:7–10 (2003).

20. Hiddleston, H. R. and others, . "Comparisons of Deformable Mirror Models and Influence Functions," *SPIE: Active and Adaptive Optical Systems*, *1542*:20–33 (1991).

21. Jenkins, C. H. and others, . "Improved Surface Accuracy of Precision Membrane Reflectors Through Adaptive Rim Control," *American Institute of Aeronautics and Astronautics*, *1983*:2302–2309 (1998).

22. Kendrick, S. E. and others, . "Optical Characterization of the Beryllium Semi-Rigid AMSD Mirror Assembly," *Proceedings of SPIE, Optical Manufacturing and Testing*, *5180*:180–187 (2003).

23. Kuo, C. "Optical Tests of an Intelligently Deformable Mirror for Space Telescope Technology," *Optical Engineering*, *33(3)*:791–800 (March 1994).

24. Maji, A. K. "Shape Measurement and Control of Deployable Membrane Structures," *Experimental Mechanics*, *Vol. 40(2)*:154–159 (June 2000).

25. Mayo, J. and others, . "Ultra-Lightweight Optics for Space Applications," *Proceedings of SPIE*, *4013*:687–698 (2000).

26. Menikoff, A. "Actuator Influence Functions of Active Mirrors," *Applied Optics*, *Vol. 30, No. 7*:833–838 (March 1991).

27. Paradies, R. and Hertwing, M. "Shape Control of Adaptive Composite Reflectors," *Composites: Part B*, *30*:65–78 (1999).

28. Piezo Systems, I., "Piezo Stack Product Sheet." Online at http://www.piezo.com/ prodstacks1.html, January 2006.

29. Romeo, R. C. and others, . "Ultra-Lightweight and Hyper-Thin Rollable Primary Mirror for Space Telescopes," *UV, Optical, and IR Space Telescopes and Instruments, Proceedings of SPIE*, *4013*:634–640 (2000).

30. Sablehaus, J. H. and others, . "An Overview of the James Webb Space Telescope (JWST) Project," *Proceedings of SPIE, SPIE, Bellingham, WA*, *5899* (2005).

31. Shepherd, M. J. and others, . "Quasi-static Optical Control of In-plane Actuated, Deformable Mirror: Experimental Comparison with Finite Element Analysis," *AIAA* (2006).

32. Sobers, D. M. *Smart Structures for Control of Optical Surfaces*. MS thesis, Air Force Institute of Technology, March 2002.

33. Sobers, D. M. and others, . "Smart structres for control of optical surfaces," *American Institute of Aeronautics and Astronautics*, 1–11 (2003).

34. Thorburn, W. G. and Kaplan, L. "A Low Voltage Electrodistortive Mirror System for Wavefront Control," *SPIE*, *1543*:52–64 (1991).

35. Trad, E. *Dynamic Characterization of Thin Deformable PVDF Mirror*. MS thesis, Air Force Institute of Technology, March 2005.

36. Tsuno, K. and others, . "NT-SiC(New-Technology Silicon Carbide): Application for Space Optics," *Proceedings of SPIE*, *5868*:1–8 (2005).

37. Wang, P. K. C. and Hadaegh, F. Y. "Modal Noninteracting Controls for Deformable Mirrors," *Second IEEE Conference on Control Applications, Vancouver, B.C.*, 121–128 (September 1993).

38. Washington, G. "Smart aperture antennas.". 801–805. June 1996.

39. Wen, Q., "IEEE-754 Floating-Point Conversion Website." Online at http://babbage.cs.qc.edu /courses/cs341/IEEE-754.html, January 2006.

40. Zang, K. and others, . "Development of X-Ray Reflectors for the Constellation-X Observatory," *Proceedings of SPIE*, *Vol. 5168*:168–179 (2004).

41. Zhu, L. and others, . "Adaptive control of a micromachined continuous-membrane deformable mirror for aberation compensation," *Applied Optics*, *Vol. 38(1)*:168–176 (January 1999).

Vita

Gina A. Peterson was raised in the suburbs of Milwaukee, WI. She graduated from Rufus King High School, Milwaukee, WI, in 1997. Upon graduation she attended the distinguished University of Minnesota, Institute of Technology. In 2001 she graduated with a Bachelor of Science in Engineering, Mechanical Engineering.

Gina was commissioned a Second Lieutenant in the United States Air Force through the Minnesota AFROTC program. She spent her first assignment at the National Air and Space Intelligence Center (NASIC) as a foreign weapons officer. She was accepted to the Air Force Institute of Technology for her second assignment, where she would pursue a Master of Science in Space Systems. Upon graduation in 2006, she will be assigned to the Space and Missile Command in L.A.

Permanent address: 2950 Hobson Way
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
21–03–2006	Master's Thesis	Sep 2004 — Mar 2006

4. TITLE AND SUBTITLE

Control Demonstration of a Thin Deformable In-Plane Actuated Mirror

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Peterson, Gina A., CPT, USAF

5d. PROJECT NUMBER

05-165

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GSS/ENY/06-M10

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFOSR
Lt. Col. Sharon Heise
4015 Wilson Blvd, Rm 713
Arlington VA, 22203-1954

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Current imaging satellites are limited in resolution and coverage area by the aperture size of their primary optical mirror. To get a large optical mirror into space, current launch weight and size restrictions must be overcome. Membrane-like optical mirrors can overcome these restrictions with their very lightweight and flexible properties. However, thin, deformable membrane mirrors are very susceptible to the space environment and require active control for surface stabilization and shaping. The primary goal of this research is to demonstrate that an in-plane actuated membrane-like deformable optical mirror can be controlled to optical wavelength tolerances in a closed-loop system. Fabrication and characterization of a five-inch membrane-like optical mirror is carried out based on efforts made in previous research. A data acquisition system to implement closed-loop feedback is characterized. Validity of a closed-loop control method is demonstrated with the in-plane actuated deformable mirror.

15. SUBJECT TERMS

Membrane-like Optical Structures, PVDF in-plane Actuation, In-Plane Actuation, Adaptive Optics

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Richard G. Cobb, Professor, AFIT/ENY
U	U	U	UU	131	19b. TELEPHONE NUMBER *(include area code)* (937) 785–3636, ext 4559